Provenance et Qualité dans les Workflows Orientés Données : application à la plateforme WebLab

Clément Caron

▶ **To cite this version:**

Clément Caron. Provenance et Qualité dans les Workflows Orientés Données : application à la plateforme WebLab. Base de données [cs.DB]. Université Pierre et Marie Curie - Paris VI, 2015. Français. NNT : 2015PA066568 . tel-01331050

HAL Id: tel-01331050

https://theses.hal.science/tel-01331050

Submitted on 13 Jun 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE DE DOCTORAT DE
l'UNIVERSITÉ PIERRE ET MARIE CURIE

Spécialité

Informatique

École Doctorale Informatique, Télécommunications et Électronique (Paris)

Présentée par

Clément Caron

Pour obtenir le grade de

DOCTEUR de l'UNIVERSITÉ PIERRE ET MARIE CURIE

Sujet de la thèse :

Provenance et Qualité dans les Workflows Orientés Données Application à la Plateforme WebLab

soutenue le 3 Novembre 2015

devant le jury composé de :

M. Bernd Amann	Directeur de thèse
M. Camelia Constantin	Co-encadrante
M. Maria-Esther Vidal	Rapporteure
M. Daniela Grigori	Rapporteure
M. Genoveva Vargas-Solar	Examinatrice
M. Christophe Marsala	Examinateur
M. Stephan Brunessaux	Invité
M. Patrick Giroux	Invité

Résumé

La plateforme Weblab est un environnement de définition et d'exécution de chaines de traitements média-mining développé par le service IPCC [1] d'Airbus Defence and Space. Il s'agit d'une plateforme ouverte qui permet l'intégration de composants logiciels externes comme des traducteurs automatiques, extracteurs d'entités nommées, etc. La richesse de composants existants permet aux concepteurs de construire des chaines média-mining très complexes, mais pose également des problèmes liés à la sensibilité de la qualité des résultats par rapport aux composants utilisés. Avant le début de cette thèse, aucun outil n'existait pour l'analyse et l'amélioration de la qualité de workflows WebLab.

La problématique principale de la thèse repose sur le fonctionnement dit *boite noire* des services WebLab (les données utilisées et créées ne sont pas connues). En effet, les COTS [2] et les services fournis par des tiers ne révèlent pas toujours le détail de leur fonctionnement. Il est alors compliqué pour un expert d'identifier les dépendances entre les services et les données, ainsi que les dépendances des données entre elles.

L'approche choisie est non-intrusive (l'impact repose essentiellement sur l'ajout et le traitement de métadonnées) : nous complétons la définition du workflow WebLab par des règles de provenance et de propagation de qualité. Les règles de provenance génèrent des liens de dépendance dits *grains-fins* entre les données et les services après l'exécution d'une chaine de traitements WebLab. Les règles de propagation de qualité profitent des liens inférés précédemment pour raisonner sur l'influence de la qualité d'une donnée utilisée par un service sur la qualité d'une donnée produite.

Les contributions apportées par cette thèse sont :

1. un modèle de génération de liens de provenance reposant sur l'utilisation de règles de dépendance entre les données, évaluées sur les documents XML issus de l'exécution d'un workflow WebLab. Ces règles sont définies comme une combinaison de requêtes XPath étendues par des variables de liens.
2. un modèle de propagation de valeurs de qualité sur un graphe de provenance, ajoutant aux fragments XML des ressources WebLab des annotations de qualité spécifiques aux types des données, et inférant de nouvelles valeurs à partir d'autres annotations de qualité et de règles de propagation de qualité.
3. une extension de l'architecture WebLab avec l'implémentation de nos deux modèles, ainsi qu'une interface utilisateur facilitant la création de règles pour les utilisateurs.

1. Information Processing Control and Cognition
2. Commercial Off-The-Shelf

Remerciements

Après la rédaction de cette thèse vient le difficile exercice des remerciements. Je ne pourrai citer toutes les personnes ayant contribué à son aboutissement. Si le lecteur ne trouve pas son nom dans les paragraphes suivants, et estime avoir eu une contribution positive, alors je le remercie pour l'aide apportée.

Tout d'abord, je tiens à remercier tous les acteurs de cette thèse : Patrick pour m'avoir proposé un stage au sein de l'équipe IPCC, et donné l'envie de réaliser cette thèse ; Stephan et Jean pour m'avoir accueilli dans leur équipe ; et enfin Bernd et Camelia pour leur encadrement sans lequel cette thèse n'aurait jamais abouti. Ces personnes ont permis à cette thèse CIFRE entre Airbus Defence & Space et le LIP6 de voir le jour, et d'évoluer jusqu'à sa complétion.

Je remercie également tous les membres du jury pour avoir porté de l'attention à mon travail : Mme Daniela Grigori et Mme Maria-Esther Vidal pour leur analyse de mon manuscrit, ainsi que Mme Genoveva Vargas-Solar et M. Christophe Marsala pour avoir accepté de faire partie du jury de cette soutenance.

Je remercie ensuite l'ensemble de l'équipe IPCC pour m'avoir soutenu tout au long de cette thèse. Leur aide sur les aspects techniques et leur enthousiasme ont fait de ces années une inoubliable entrée dans la vie active. Mes remerciements vont également à l'équipe BD du LIP6 pour leur accueil, m'offrant un aperçu du milieu de la Recherche qui m'intriguait tant. Je remercie de plus André pour toutes les idées apportées durant ses séjours hivernaux.

Je remercie tous mes co-doctorants, trop nombreux pour être cités, pour avoir su me rassurer dans les moments d'incertitude.

Je remercie également l'ensemble du personnel administratif pour leur aide dans toutes les démarches : Véro et Marie-Angèle chez Airbus, Ghislaine et Nadine au LIP6.

Je remercie bien entendu ma famille : Enigma et Sherlock pour avoir fait de moi qui je suis aujourd'hui, mais également mes frères Austin et Joker qui ont su me montrer tant de moyens de s'amuser.

Enfin, mes derniers remerciements vont à Catwoman qui a décidé de m'accompagner dans de futures aventures, et qui m'a tant soutenu dans la clôture de mon manuscrit, ainsi qu'à Yoshi pour apporter tant de fantaisie dans notre vie.

Table des matières

Table des matières **iv**

Table des figures **vi**

1 Introduction **1**
1.1 Motivation . . . 1
1.1.1 Media-mining . . . 1
1.1.2 La plateforme WebLab . . . 1
1.2 Problématique . . . 4
1.3 Approche . . . 5
1.4 Principaux résultats de la thèse . . . 6
1.4.1 Modèle de Provenance WebLab . . . 6
1.4.2 Modèle de Qualité WebLab . . . 6
1.4.3 Réalisation . . . 6
1.5 Plan de la thèse . . . 7

2 État de l'art **9**
2.1 Qualité des données . . . 9
2.1.1 Les dimensions de qualité . . . 9
2.2 Améliorer la qualité des données . . . 14
2.2.1 Qualité source et dérivée . . . 14
2.2.2 Réparations de données et de workflows . . . 15
2.3 Provenance des données et Workflows . . . 18
2.3.1 Graphe de Provenance . . . 18
2.3.2 Utilisation de Provenance . . . 19
2.3.3 Standards . . . 21
2.3.4 Systèmes de workflows et provenance . . . 23
2.3.5 Provenance pourquoi/comment/où . . . 25
2.3.6 Granularité de la Provenance . . . 28
2.3.7 Génération de la provenance . . . 30
2.4 Conclusion . . . 33

3 Modèle de provenance **35**
3.1 Documents WebLab . . . 35
3.2 Modèle de provenance WebLab . . . 37

3.2.1 Graphe de provenance WebLab 37
3.2.2 Génération du graphe structurel 40
3.2.3 Sémantique structurelle 44
3.2.4 Évaluation des règles 45
3.2.5 Sémantique temporelle 47
3.3 Extensions 48
3.4 Règles contextuelles 50
3.5 Conclusion 52

4 Gestion de la qualité **53**
4.1 Modèle d'annotation de qualité 53
4.2 Modèle d'inférence de qualité 58
4.3 Conclusion 63

5 Réalisation **65**
5.1 WebLab-PROV 65
5.1.1 Modèle d'enregistrement 66
5.1.2 Enregistrement des traces d'exécution 69
5.1.3 Génération du graphe de provenance 70
5.1.4 Traducteur de règles 73
5.2 WebLab-Quality 75
5.3 Architecture globale 80
5.4 Interface utilisateur 82
5.5 Conclusion 84

6 Conclusion **85**
6.1 Résumé 85
6.1.1 Modèle de Provenance WebLab 85
6.1.2 Modèle de Qualité WebLab 85
6.1.3 Réalisation 86
6.2 Perspectives 86

Bibliographie **87**

Table des figures

1.1 Chaine de traitements media-mining 2
1.2 Exemple de document WebLab 3

2.1 Extrait de poème de Verlaine 10
2.2 Catalogue géographique 11
2.3 Entités extraites 11
2.4 Graphe de provenance 19
2.5 Vue des clusters au sein d'un graphe de provenance 20
2.6 IHM Taverna (https ://en.wikipedia.org/wiki/Apache_Taverna) 23
2.7 IHM Kepler (https ://kepler-project.org/developers/interest-groups/provenance-interest-group/interfaces-to-provenance-for-reap) 24
2.8 IHM Vis-Trails (http ://www.aosabook.org/en/vistrails.html) 25
2.9 Tableaux de données 26
2.10 Service black-box 29
2.11 Service grey-box 30
2.12 Service white-box 30
2.13 Graphe de provenance : Peluche 31
2.14 Provenance prospective : exemple 31
2.15 Provenance par inversion : exemple 32
2.16 Provenance rétrospective : exemple 32

3.1 Exemple de document WebLab 35
3.2 Exemple de document WebLab avec annotation temporelle 37
3.3 Graphe de provenance 38
3.4 Exemple de document WebLab 42
3.5 Application des patrons XPath 46
3.6 Jointure des patrons XPath 46
3.7 Extrait du document 46
3.8 Règle sans variable 47
3.9 Fonctions de Skolem 50

4.1 Exemple de dimensions de qualité 54
4.2 Annotations de qualité A 56
4.3 Valeur(A) 57
4.4 Exemple d'incohérence 57

4.5 liens(M,L) 60

5.1 Architecture globale 65
5.2 WebLab-PROV : architecture fonctionnelle 66
5.3 Modèle RDF 66
5.4 Exemple d'enregistrement RDF : Activity 67
5.5 Exemple d'enregistrement RDF : Agent 67
5.6 Exemple d'enregistrement RDF : Entity 68
5.7 Documents d'entrée et de sortie 69
5.8 Document final 71
5.9 Document annoté 71
5.10 Extrait de grammaire JavaCC 74
5.11 Grammaire WebLab-PROV 75
5.12 Récapitulatif des règles du traducteur 77
5.13 Exemple application de qualité 78
5.14 Exemple RDF 79
5.15 Architecture globale 81
5.16 Explorateur de provenance 82
5.17 Vue globale de l'application 82

1 Introduction

1.1 Motivation

1.1.1 Media-mining

Le développement d'Internet et la numérisation massive de documents de tous types ont amené le besoin d'analyser ces contenus, ouvrant la voie aux applications media-mining. Le processus standard d'une telle application suit 3 phases : l'extraction du contenu, l'analyse des informations, et enfin l'indexation des données pertinentes. Cependant, la complexité de ces traitements est très variable, allant de simple pour l'analyse de sites internet en anglais, à très compliqué pour l'extraction d'informations de flux vidéo en chinois. Cette complexité variable se reflète alors sur les composants utilisés, ainsi que sur le workflow en lui-même. Pour notre premier cas d'utilisation, l'analyse du texte demandera une normalisation et une extraction des entités pertinentes. Pour le second cas, la vidéo devra être démultiplexée (séparation des images et de l'audio) : les images pourront être analysées pour reconnaitre des personnes, pendant que l'audio pourra être transcrit en texte afin d'en extraire des entités pertinentes. Les traitements utilisés dans le media-mining peuvent être couteux en termes de performance et d'argent, sans pour autant garantir une qualité satisfaisante.

1.1.2 La plateforme WebLab

Dans une démarche de simplification des outils media-mining, le service IPCC de Airbus Defence and Space a développé la plateforme WebLab. La plateforme possède une architecture orientée services (SOA), définissant chaque composant à l'aide du standard *WSDL*[1] et communiquant avec celui-ci à l'aide de *SOAP*[2]. Cette approche permet d'envisager chaque composant comme un fournisseur de service(s), et le traitement d'un ensemble de documents multimédias pour en extraire des informations ou de la connaissance se traduit par une orchestration de différents services. On parle alors d'une chaine de traitements de document pour désigner cette orchestration des services. Ainsi, au sein de la plateforme WebLab, l'analyse de documents se fait à l'aide de chaine de traitements (ou workflow), exécutant des composants les uns à la suite des autres, comme illustré sur la figure 1.1 ci-dessous. Un document XML transite entre chaque service au sein duquel les informations sont stockées.

1. Web Services Description Language
2. Simple Object Access Protocol

Cette orchestration de services prend une vidéo en entrée et extrait les visages et textes reconnus sur les images, et les entités nommées (places ou personnes connues) identifiées sur la piste audio. Ce workflow est une simplification d'une chaine de traitements proposée dans le projet européen AXES, ayant pour objectif la création d'une base d'archives vidéo interrogeable à partir de mots-clés tels que le nom d'une personne présente ou les thèmes évoqués dans une vidéo.

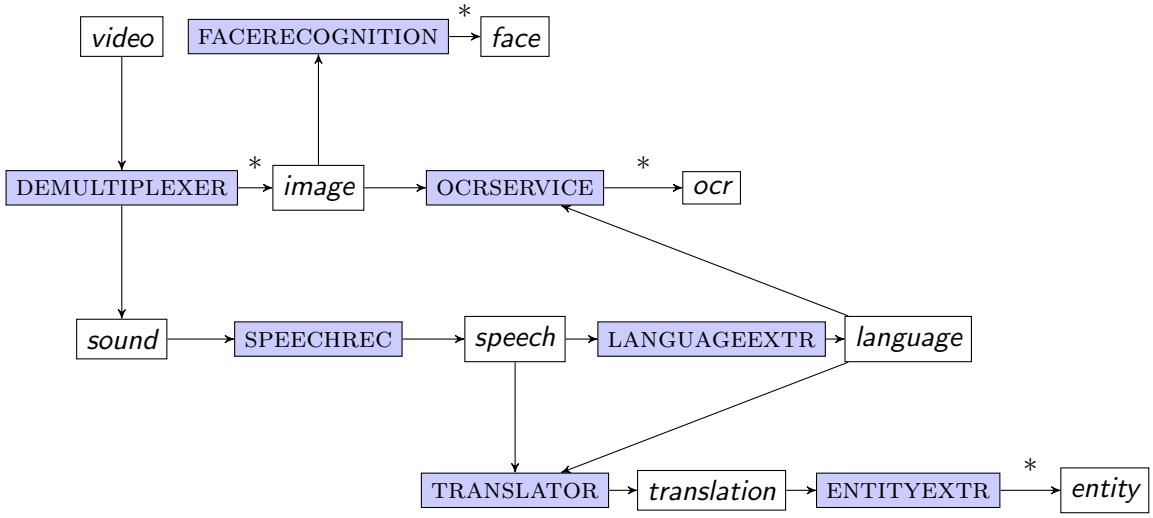


FIGURE 1.1 – Chaine de traitements media-mining

Dans un premier temps, la vidéo est démultiplexée. Cette opération consiste à séparer la piste audio des images. Ce traitement fait évoluer le document WebLab en créant un nœud AudioMediaUnit pour la piste audio, et un nœud ImageMediaUnit pour chaque image extraite. Ces nœuds sont visibles sur le document WebLab de la figure 1.2. Par la suite, deux traitements s'effectuent en parallèle : celui de la piste audio, et celui des images.

La piste audio, nœud AudioMediaUnit, est transformée en texte par le transcripteur, créant le premier nœud TextMediaUnit sous le même parent *Resource* (figure 1.2). La transcription d'un texte consiste à mettre à l'écrit les paroles des interlocuteurs ou des voix off présentes sur la piste audio, avec si possible une différenciation des personnes par rapport à la voix pour ne pas perdre le lecteur (différents outils permettent de faire ce travail, payants ou gratuits, comme *Sphinx* par exemple).

La langue est ensuite extraite à partir du sous-nœud *Content* (contenant la version texte de la piste audio), créant le sous-nœud *Language* au sein de l'unité de texte. Plusieurs méthodes existent pour faire ce type de traitement comme, par exemple,

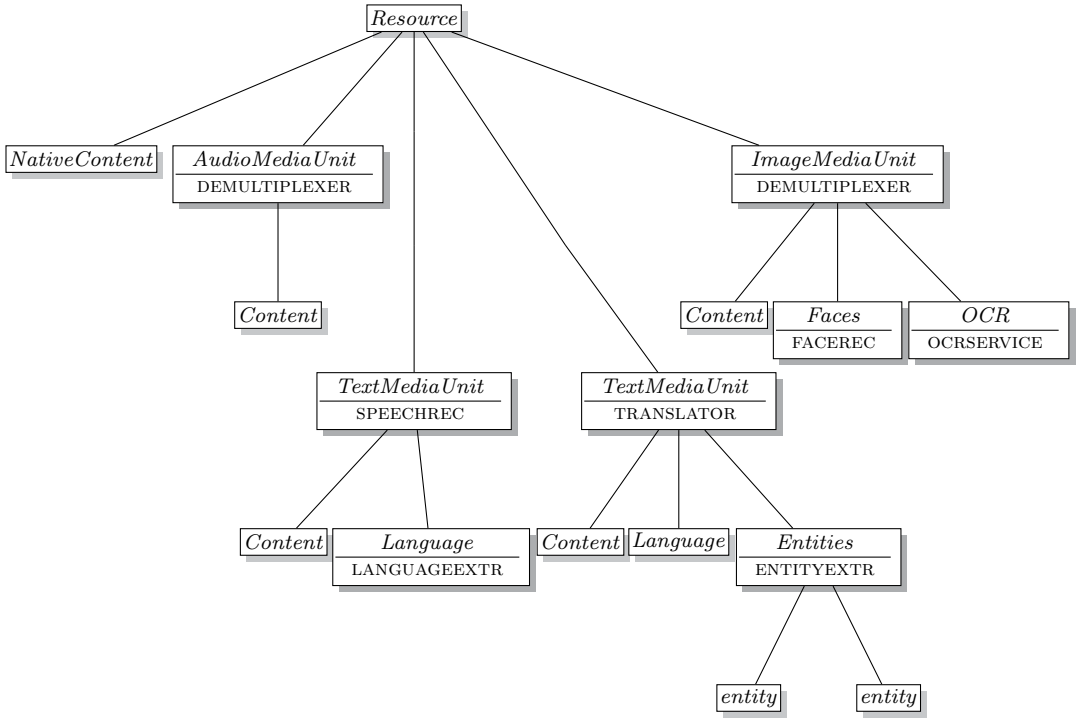


FIGURE 1.2 – Exemple de document WebLab

l'utilisation de n-grammes, permettant d'identifier la langue par rapport au nombre d'apparitions de chaque lettre dans un texte.

L'entité TextMediaUnit est ensuite traduite depuis la langue identifiée vers une nouvelle plus facile à analyser. Le traducteur crée un nouveau fils TextMediaUnit du nœud *Resource*, contenant le texte traduit (*Content*) et annoté avec la langue cible. Différents composants de traduction peuvent être utilisés, comme ceux proposés par Google ou Microsoft (Bing), et il existe des solutions gratuites comme *Moses*.

Les entités nommées sont ensuite extraites de ce texte, créant un nœud *Entity* pour chacune d'entre elles. Différents outils peuvent être utilisés, des plus simples fondés sur une liste de mots-clés, jusqu'aux plus puissants comme *Gate* permettant l'analyse linguistique de phrases pour identifier des personnes, des lieux ou encore des évènements.

Les images sont traitées séparément par deux services, dont le premier effectue l'analyse des visages visibles sur chaque image. L'analyse se fait image par image, en étudiant les écartements entre différents points clés des visages, comme celui des yeux.

Le second traitement sur les images est la reconnaissance de texte ou OCR. Ce traitement est synchronisé avec le service d'extraction des langues car il utilise celle

identifiée dans le texte pour faciliter la reconnaissance des mots dans les images. Les nœuds ainsi créés se retrouvent enfants de chaque *ImageMediaUnit*.

Au cours de ce manuscrit, nous utiliserons cet exemple de traitement media-mining.

1.2 Problématique

La qualité des données résultantes de l'exécution d'un workflow media-mining est très variable. En effet, la qualité inégale des données sources, associée à une complexité des traitements d'analyse très sensibles à la qualité des données traitées font que le résultat final n'a pas toujours la qualité souhaitée par les utilisateurs. Dans le cadre de la plateforme WebLab, le document final contient à la fois les résultats finaux et les résultats intermédiaires, mais l'expert ne possède pas suffisamment d'informations pour relier les données entre elles, et en particulier l'influence d'un résultat intermédiaire sur le résultat final. L'expert doit alors remplacer ou modifier les composants de manière empirique afin d'espérer améliorer la qualité des workflows.

Dans ce contexte, l'utilisation d'un système d'évaluation de qualité afin de noter puis recommander un service ou une composition de services est nécessaire. La mise en place d'un tel concept associé aux travaux de génération automatique de chaines de services permettrait la création de chaines à partir d'éléments ou compositions d'éléments évalués comme étant le meilleur choix pour répondre à une demande précise et dans un contexte précis.

Il est nécessaire de définir formellement ce qu'est la qualité d'un service, ainsi que les moyens de l'évaluer. Sachant que le fonctionnement des services utilisés au sein de la plateforme WebLab n'est pas toujours connu, l'analyse de leur qualité soulève la problématique suivante : comment qualifier un service sans connaitre son fonctionnement ?

Les travaux existants sur la qualité de service (QoS) ne prennent en compte que les qualités dites *non fonctionnelles* (prix, temps moyen de traitement, disponibilité, niveau de sécurité, etc.), il nous faut donc trouver un autre moyen afin d'évaluer la qualité *fonctionnelle* (précision, complétude, etc.) discriminant les traitements effectués par les composants.

Ce type de qualité est davantage présent dans les traitements orientés données. Notre principale hypothèse est que la qualité fonctionnelle d'un service dépend des données auxquelles le service est appliqué. Par exemple, l'utilisation d'un traducteur anglais-français fonctionne très bien si le texte est anglais, mais très mal pour

toute autre langue. Pour modéliser cette interaction entre données et services, il est nécessaire d'identifier les données utilisées et créées par chaque service. Plus précisément, il faut identifier quel sous-ensemble de données a été utilisé par le service pour créer un autre sous-ensemble de données. Une fois ce problème résolu, il sera possible de qualifier les données liées à un service.

Pour faciliter la mise en place de notre système de gestion de qualité au sein de la plateforme WebLab, celui-ci doit pouvoir s'intégrer en effectuant un minimum de modifications, en particulier concernant le temps d'exécution des chaines de traitements.

1.3 Approche

Pour modéliser l'interaction entre données et services, nous avons d'abord défini un modèle de provenance adapté aux workflows WebLab. Ce modèle permet de représenter d'une manière explicite les informations concernant l'utilisation et la génération de données par des services. Notre objectif était d'étendre la plateforme sans changer le fonctionnement des services et des modules existants. Afin de définir et générer les liens de provenance, nous avons construit un système de règles associées à chaque composant, permettant d'identifier au sein d'un document WebLab les données utilisées et créées. Ces règles permettent de construire, a posteriori de l'exécution d'un workflow, un graphe de provenance comprenant l'ensemble des données.

Ensuite, nous avons développé un modèle de qualité permettant de lier différentes dimensions de la qualité entre les nœuds du graphe de provenance. Ce modèle associe des règles de propagation de la qualité aux règles de provenance des services, dans un but de fonctionnement semi-automatique : l'utilisateur devra spécifier si le résultat et/ou les données intermédiaires sont corrects ou non. En effet, la qualité étant un concept subjectif, il n'est pas possible d'avoir un modèle totalement automatique.

Ces deux modèles ont été implémentés afin de prouver leur efficacité. Le prototype permet à l'utilisateur de créer les règles de provenance associées à un service, avec une aide à la création par la sélection de nœuds dans le document WebLab. Le graphe de provenance généré grâce aux règles créées est ensuite stocké dans un triplestore. Au sein de celui-ci, les règles de qualité, formulées sous forme de règles d'inférence, évaluent la qualité de chaque nœud à chaque modification apportée au triplestore.

1.4 Principaux résultats de la thèse

Dans cette section, nous présentons les principaux résultats obtenus au cours de nos travaux, en commençant par notre modèle de provenance, puis notre modèle de qualité, et nous terminons avec la description de nos implémentations effectuées.

1.4.1 Modèle de Provenance WebLab

Nous proposons un modèle d'annotation de règles de provenance associées aux services WebLab permettant de générer les liens de dépendances entre les nœuds XML des documents WebLab. Ces règles sont décrites sous un format inspiré de XPath, et sont assez génériques pour être intégrées dans d'autres systèmes de workflows fondés sur l'échange de documents XML. Ce modèle ne demande aucune modification des services web exécutés, et est très peu invasif concernant l'orchestrateur de service : le modèle enregistre quelques métadonnées telles que la date d'exécution, permettant d'éviter les incohérences dans le graphe.

1.4.2 Modèle de Qualité WebLab

Nous avons par la suite développé un modèle de règles de qualité associées aux règles de provenance créées précédemment. Ces règles viennent en complément des outils d'analyse de la qualité existants (comme un analyseur orthographique par exemple), très dépendants du type des données. Ces règles relient une ou plusieurs entrées avec une ou plusieurs sorties à l'aide d'opérateurs de comparaison, permettant d'effectuer des liens tels que : la *cohérence* du texte en entrée du service est toujours supérieure à la *cohérence* du texte en sortie. Elles permettent ainsi d'estimer la qualité d'une donnée, mais également de détecter des services qui ne fonctionnent pas comme prévu.

1.4.3 Réalisation

Nous avons implanté ces deux modèles dans une démarche d'intégration à la plateforme WebLab. Nous avons modifié l'orchestrateur de services (Java) afin de récolter des métadonnées d'exécution, et créé une grammaire (JavaCC) permettant de transformer nos règles en un langage intelligible par la machine (XQuery). Les données de provenance sont alors enregistrées dans un triplestore. Dans celui-ci, nos règles de qualité, sous forme de règles d'inférence (Jena), permettent alors de créer les liens entre les dimensions de qualité. Nous avons également réalisé une interface graphique facilitant à l'utilisateur la création de règles de provenance.

1.5 Plan de la thèse

Dans le chapitre 2, nous présentons un état de l'art des différents domaines de recherche liés à notre problématique. Dans un premier temps, nous présentons les différentes dimensions de qualité, puis nous précisons comment il est possible de l'améliorer. Dans un second temps, nous introduisons le domaine de la provenance, son utilisation, la notion de graphe ainsi que différents types de provenance *why*, *where*, *how*, et enfin sa génération.

Dans le chapitre 3, nous proposons un modèle de provenance applicable aux chaines de traitements de la plateforme WebLab, et plus généralement aux workflows utilisant des documents XML. Puis, dans le chapitre 4, nous y associons un modèle de qualité afin d'annoter et de déduire des valeurs de qualité des données présentes sur le graphe de provenance.

Dans le chapitre 5, nous présentons une implémentation de nos modèles montrant la faisabilité et l'intérêt de leur utilisation au sein de la plateforme WebLab.

Dans le chapitre 6, nous concluons ce manuscrit en résumant nos travaux et proposant de nouvelles perspectives de recherche.

2

État de l'art

Afin de mieux comprendre la problématique de gestion de la qualité dans le domaine des chaines de traitements media-mining, nous nous sommes intéressés aux solutions existantes dans les domaines de la *qualité* et de la *provenance des données*.

Ce chapitre est organisé en trois sections : la Qualité des données dans un premier temps (section 2.1), puis les moyens de l'améliorer (section 2.2), et enfin la Provenance (section 2.3).

Dans la première section, nous décrirons la notion de qualité de données dans les bases de données et les workflows. La qualité est souvent liée à une perception subjective et dépend du contexte d'utilisation, mais également de l'utilisateur. Il est ainsi important de la définir au plus près du besoin d'une application.

Dans la deuxième section, nous définirons les notions de provenance, ainsi que les règles de bases pour construire un modèle de provenance.

2.1 Qualité des données

Dans le contexte d'un workflow WebLab, les données que nous souhaitons évaluer correspondent aux informations d'entrée et de sortie des services. Ainsi, la qualité du résultat d'un traitement media-mining dépend de celle des services exécutés et de celle du document initial.

L'analyse du fonctionnement d'un service n'est pas toujours possible, en particulier dans le domaine de la fouille de médias, où les services sont souvent très complexes et considérés comme des boites noires. Nous avons donc orienté notre approche vers l'analyse de la qualité des informations entrantes et sortantes des services afin de pouvoir en déduire une mesure de qualité du service (si la qualité d'une entrée est bonne mais que celle du résultat est mauvaise, alors nous pouvons en déduire que le service est de mauvaise qualité ou inadapté à la tâche).

Dans cette section, nous allons étudier comment la qualité d'une information peut être évaluée, et quels sont les principaux éléments qui l'influencent.

2.1.1 Les dimensions de qualité

Mesurer la qualité d'une information est complexe. La qualité est souvent, et en particulier dans le domaine des médias, une notion reposant sur de multiples facteurs [32]. Elle peut varier selon le type d'application (veille économique, d'opinion, ou encore militaire), de données (texte non structuré, base de données, images, vidéos,

etc.), ou même selon l'utilisateur. Il est donc important d'adapter la méthode de mesure au contexte d'utilisation.

La qualité d'une information peut être décrite sur plusieurs dimensions. Chaque dimension de qualité est mesurable par une analyse automatique ou humaine. Une dimension de qualité peut être ignorée ou mise en avant selon le contexte, et ainsi la qualité globale d'une donnée s'obtient par agrégation du sous-ensemble de dimensions appropriées au contexte.

Les dimensions de qualité peuvent se référer à l'*extension* de l'information, c'est-à-dire sa valeur, ou à l'*intention* de l'information, c'est à dire son schéma [9]. Une valeur de faible qualité aura de fortes conséquences sur la qualité du résultat final, tandis qu'un schéma de mauvaise qualité pourra engendrer une anomalie du système : redondance des informations, oubli, mauvaise interprétation des données. Le modèle de gestion des informations WebLab impose un schéma qui n'est pas modifiable et nous allons donc nous concentrer sur la qualité des valeurs.

L'exemple 1 illustre cette problématique de qualité des données par un exemple d'extraction d'informations dans le poème de Verlaine *"Paris"* par un *extracteur d'entités nommées géographiques*.

Nous évaluerons la qualité des informations selon leur *précision*, *complétude* et/ou *cohérence*. Ces trois dimensions de qualité sont les plus communément utilisées pour qualifier différents types d'information [9].

Exemple 1

Soit le texte ci-dessous, issu du poème de Verlaine "Paris" (figure 2.1).

Paris n'a de beauté qu'en son histoire,
Mais cette histoire est belle tellement !
La **Seine** est encaissée absurdement,
Mais son vert clair à lui seul vaut la gloire.

Paris n'a de gaîté que son bagout,
Mais ce bagout, **encor** qu'assez immonde,
Il fait le tour des langages du monde,
Salant un peu ce trop fade ragoût.

Paris n'a de sagesse que le sombre
Flux de son peuple et de ses factions,
Alors qu'il fait des révolutions
Avec l'Ordre embusqué dans la pénombre.

FIGURE 2.1 – Extrait de poème de Verlaine

*Considérons en plus de ce texte un tableau de données géographiques (figure 2.2). Ce tableau répertorie un ensemble d'entités géographiques utilisées par différents composants WebLab avec leur nom, pays, population et la langue parlée. Une erreur typographique (Ét**e**ts-Unis) a été faite dans le tableau volontairement.*

Identifiant	Nom	Pays	Population	Langue
SeineMaritime	Seine Maritime	France	1 251 282	Française
ParisUS	Paris	Ét**e**ts-Unis	24 912	Anglaise
Paris	Paris	France	2 240 621	Française

FIGURE 2.2 – Catalogue géographique

Nous considérons finalement un extracteur d'entités nommées utilisant le tableau ci-dessus afin d'identifier les informations géographiques présentes dans le texte de Verlaine. Le service parcourt le texte à la recherche d'entités répertoriées dans son catalogue géographique et identifie les lieux présents. Un résultat possible de cette exécution de service est visible sur le tableau 2.3.

	Identifiant	Entité	Position dans le texte	
E_1 :	SeineMaritime	Seine	L3C3	✘
E_2 :	Paris	Paris	L1C1	✓
E_3 :	ParisUS	Paris	L5C1	✘
E_4 :	ParisUS	Paris	L9C1	✘

FIGURE 2.3 – Entités extraites

Nous pouvons voir ici que le service a extrait deux entités présentes dans le texte, la Seine et Paris, mais que ces deux entités ont été extraites maladroitement. En effet, l'extracteur a estimé que le département dont le nom commence par le mot Seine, et le fleuve du même nom, sont identiques. Le service a également confondu Paris, capitale de la France, avec Paris au Texas pour E_3 et E_4.

Nous allons décrire dans la suite différentes dimensions de qualité pour caractériser la qualité d'une information. Nous nous concentrerons sur les principales dimensions utilisées dans le cadre du traitement automatique des médias.

Cohérence

La cohérence peut être caractérisée par la quantité de violations de règles sémantiques définies sur une donnée ou un ensemble de données [9]. Au sein de bases de

données relationnelles, les contraintes d'intégrité représentent une instantiation de ce type de règle. Les contraintes grammaticales sont un exemple d'utilisation de la cohérence pour qualifier un texte comme le poème de Verlaine de l'exemple 1.

La mesure de la cohérence d'une information se fait à l'aide d'un rapport entre le nombre de contraintes respectées (ou l'inverse du nombre de contraintes violées) sur l'ensemble des contraintes liées à cette information.

Exemple 2

Prenons par exemple le champ Population dans la table des entités de l'exemple 1. La valeur attendue par ce champ est un nombre entier positif. Un exemple d'incohérence serait l'utilisation d'un type de donnée différent de celui initialement prévu (comme une chaine de caractères, ou encore un nombre négatif). Dans notre exemple, l'entité E_2 spécifie que la Seine a une population de 1 251 282 individus alors qu'il s'agit d'un fleuve, type de donnée où la mesure de la population humaine n'est pas applicable. L'incohérence n'est pas présente dans le tableau d'origine 2.2, mais est générée par le service d'extraction d'entités nommées en identifiant par erreur la Seine.

Un autre exemple, dans le traitement de données de type média, est la cohérence dans les textes. Elle est plus difficile à évaluer, et peut aller de la simple erreur grammaticale jusqu'à la suite de phrases vides de sens sans lien entre elles. La mesure est plus complexe, mais il existe cependant des outils d'évaluation de la qualité d'écriture (évaluation orthographique par exemple [8][24]). Un exemple est la détection de fautes grammaticales ou syntaxiques comme dans le texte de Verlaine, où le mot "encor" peut être identifié comme une erreur.

Complétude

La complétude possède diverses définitions [48] [43]. Nous retiendrons ici celle qui correspond au rapport entre la quantité de valeurs correctes répertoriées et la quantité de valeurs correctes en tout. La difficulté de cette mesure est que la quantité totale de valeurs correctes n'est souvent pas connue, et doit être estimée par l'utilisateur.

Exemple 3

En reprenant la table des entités précédente, une mesure de complétude correspond au rapport entre le nombre de champs renseignés correctement dans cette table et le nombre total de champs géographiques présents dans le poème. Ainsi, dans notre exemple, nous pouvons considérer que les éléments attendus sont les trois villes de

Paris en France, nous pouvons donc évaluer la complétude à 33% (seule une des villes ayant été localisée en France).

Une autre mesure de complétude existe pour les données (ou documents) sous forme de flux. De nouvelles parties de textes sont créées et traitées au fur et à mesure du temps, faisant évoluer le nombre idéal d'entités à extraire, hors la complétude est une caractéristique statique. Dans ce cas, la notion de complétabilité [42] est ajoutée au problème. L'idée ici est que la complétude maximale de référence évolue avec le temps, imposant donc réévaluer la complétude à chaque instant.

Précision

La précision peut être définie comme une distance entre un élément et la valeur attendue. Elle peut mesurer des distances binaires (correct / incorrect), ou numériques (nombre de lettres de différence dans une distance syntaxique) [9].

Exemple 4

*Si nous reprenons notre exemple, la valeur Ét**e**ts-Unis du champ Pays de la ligne E_2 est syntaxiquement **incorrecte** (Ét**e**ts-Unis au lieu de Ét**a**ts-Unis). La précision peut être évaluée de manière binaire (faux) ou numérique en calculant le nombre d'insertions, suppressions ou modifications de caractères avant d'avoir une réponse attendue (distance de Levenshtein [37]). La précision numérique de cette donnée est **1** (une unique modification de caractère par rapport à la valeur correcte).*

La précision peut également être mesurée par la distance sémantique entre deux éléments en utilisant la distance de Wu/Palmer [49] par exemple. Cette distance se calcule à l'aide de l'expression

$$\frac{2D}{D_1 + D_2}$$

où D_1 et D_2 sont les profondeurs dans l'ontologie des deux entités comparées, et D est la profondeur de leur premier ancêtre commun. Cette distance permet d'identifier un degré de similarité entre deux entités, par exemple deux villes différentes auront un degré de similarité plus élevé qu'une ville et une voiture.

Les principales dimensions que nous utilisons dans nos exemples sont celles listées ci-dessus. Il en existe de nombreuses autres telles que l'*accessibilité* [1], évaluant la possibilité pour les utilisateurs de comprendre les informations (langues disponibles, simplicité d'écriture), ou la *qualité des sources d'informations* [48], dimension

répertoriant la confiance et la crédibilité que l'on peut donner à une source d'information. Ces dimensions peuvent être plus ou moins objectives, et nécessiter une intervention humaine.

Néanmoins, nous verrons dans le chapitre 4 que notre modèle de qualité est assez générique pour représenter et traiter toutes ces mesures.

2.2 Améliorer la qualité des données

2.2.1 Qualité source et dérivée

Nous considérons deux types de données : les données créées par des composants sources (sans donnée d'entrée), et les données issues du traitement des données d'entrées.

Définition 1 *Données sources et dérivées*

*Nous appelons donnée **source** toute donnée en entrée d'une chaine de traitements, ou créée par un composant sans dépendance vers une autre donnée.*

*Nous appelons donnée **dérivée** toute donnée créée par un composant à partir d'une ou plusieurs autres données (sources ou dérivées).*

Exemple 5

*Considérons le passage du poème de Verlaine traité par l'extracteur d'entités nommées. Ici le texte est une donnée en entrée de traitement, sans dépendance avec une autre donnée (donnée **source**).*

Le catalogue d'entités géographiques utilisé par l'extracteur d'entités nommées ne dépend d'aucune donnée et est également une donnée source.

*Le tableau d'entités extraites quant à lui est issu de l'exécution de l'extracteur d'entités nommées sur le texte. Son contenu est considéré comme **dérivé** du poème de Verlaine et du catalogue d'entités géographiques.*

*L'extraction d'entités nommées utilise généralement la formulation des phrases pour détecter les identifications de lieux, nous pouvons considérer que la **précision** de chaque entité dépend de la **cohérence** du texte de Verlaine.*

La qualité des données **sources** dépend uniquement de son composant source (qui peut être un humain ou un extracteur automatique). La qualité des données **dérivées**, mauvaise ou bonne, est dépendante des données d'entrée des services ou des services les ayant créées. Cette dépendance entre les informations (ou provenance) joue un rôle clef dans la qualité de l'information dans les applications de traitements

automatiques des données car elle permet de raisonner sur l'origine et la propagation d'erreurs dans le workflow.

En effet, l'identification de la *Seine* par le service en tant que lieu est erronée, la précision de cette information est donc considérée comme *faible*. Par connaissance des liens entre les données (ou provenance), il est possible de postuler que la qualité du service d'extraction d'entités nommées, ou la **cohérence** du texte dont le terme Seine est issu, est faible. Sachant que la cohérence du texte est bonne, nous pouvons en déduire qu'il y a un problème avec le service d'extraction d'entités nommées, et qu'il faut changer ce composant pour de futurs traitements.

De manière générale et pour les graphes de données plus complexes, la détection d'une qualité faible pour une donnée peut être répercutée sur les sources dont elle dépend afin d'identifier les données ou les traitements de mauvaise qualité. Dans le chapitre 3 (page 35), nous allons montrer un modèle de provenance facilitant la détection d'erreurs dans un workflow.

2.2.2 Réparations de données et de workflows

L'amélioration de la qualité de données issues d'un workflow peut suivre deux approches : la réparation des données actuelles, ou l'apprentissage pour les exécutions futures des mêmes traitements.

Réparation des données (Database Curation) Dans le domaine des bases de données, le problème de la réparation de données (*database curation* [13]) est un domaine de recherche à part entière. L'idée ici est, connaissant les données sources et les traitements, de recalculer les données dérivées pour corriger les erreurs, ou confirmer que les erreurs se situent dans les sources.

La réparation de données est née dans un contexte de capitalisation de données importées de sources extérieures, plus ou moins fiables. Ces données sont susceptibles d'avoir subi des modifications ou annotations, manuelles ou automatiques, ou encore d'être incomplètes [36] [29] et peuvent donc nécessiter d'être *réparées*. L'objectif second dans ce domaine est de catégoriser les données afin de faciliter leur recherche.

Les utilisateurs des outils de réparation de données attachent une grande importance à la qualité [3], la confiance [30] [33], et à la fiabilité [11] [2] [21] des données. Dans tous les cas, connaitre l'origine de l'information est un facteur majeur pour l'évaluation de la confiance apportée à une donnée. D'autres travaux [19] [10] permettent de répondre à différents problèmes dans le contexte de bases de données

partagées, comme la confiance portée à une collection de données distantes, ou encore l'amélioration du stockage par suppression de la redondance (telle les doublons dans une base de données).

Buneman [13] démontre que l'utilisation d'informations de provenance permet d'assurer l'intégrité des données et propose un modèle de provenance pour ce type de système, allant jusqu'à référencer toutes les modifications utilisateurs. Nous verrons dans la section 2.3 les différents moyens d'extraction de la provenance dans un workflow.

Réparation de Workflow Dans le domaine des orchestrations de services, l'objectif n'est pas de réparer les données, mais d'aider l'expert à identifier les composants sensibles afin d'améliorer les traitements futurs.

Les deux moyens d'améliorer les données de sortie d'une chaine de traitements sont soit d'améliorer les composants qui la constituent, soit d'améliorer la qualité des données sources. Ce choix dépend de la volonté des utilisateurs, ainsi que des possibilités offertes par le workflow. En effet, certains composants fonctionnent avec un type précis de données d'entrée, comme par exemple les traducteurs ne fonctionnant que pour une langue donnée. Dans le cas de la traduction, l'utilisateur devra changer au choix le traducteur ou la langue du texte d'entrée.

Différents travaux de recherche dans le domaine de l'orchestration de services ont vu le jour [51][52][53]. Les travaux sur la qualité de services (QoS) traitent de problèmes tels que la sélection automatique de services à partir de contraintes [50], ou le classement des services sur plusieurs critères [16] [38] [28] (prix, temps de traitement, ou compatibilité avec d'autres services).

Dans le contexte des services web, chaque composant est caractérisé par deux types d'information : les propriétés *fonctionnelles* et *non-fonctionnelles*. Les propriétés non-fonctionnelles sont les métadonnées d'un composant. Cela peut être son prix d'utilisation (dans le cas d'un service payant), son temps d'exécution moyen, sa fiabilité (pourcentage de succès d'exécution ou de disponibilité), etc. Les données fonctionnelles correspondent aux données liées au fonctionnement intrinsèque d'un composant. La gestion de la qualité de ce type de composant est complexe, en particulier lorsque le fonctionnement du service est inconnu (service black-box). Dans ce type de cas, l'étude des données d'entrée et de sortie permet d'évaluer la qualité de manière similaire au cas des bases de données relationnelles, avec les notions de dimensions de qualité et de types de données.

La gestion de la qualité des données fonctionnelles permet de sélectionner dynamiquement un composant suivant la qualité des données d'entrées, ou suivant une valeur de qualité particulière. L'idée est, par exemple, de choisir le traducteur le plus adéquat pour une donnée dont la langue est renseignée comme valeur de qualité.

La connaissance de la provenance des données dans un système permet l'évaluation de la qualité des informations. Nous nous intéressons donc par la suite à l'évaluation de la provenance dans un système orienté services.

2.3 Provenance des données et Workflows

Nous avons vu dans la section précédente l'importance de la provenance sur l'évaluation et l'amélioration de la qualité des données dans un workflow orienté données. Cette section est consacrée à la gestion de la provenance dans les systèmes d'information.

Nous allons commencer par donner différents cas d'utilisation de la provenance, puis nous définirons la notion de graphe de provenance, avant de montrer différents types de provenances, ainsi que les modèles de gestion de provenance existants.

2.3.1 Graphe de Provenance

La provenance dans un système de traitement de l'information [15] [39] est représentée sous forme d'un graphe que nous appelons graphe de provenance [26]. Nous le définissons ainsi :

Définition 2 *Graphe de provenance*

*Un **graphe de provenance** est un triplet (S, D, E) où S est un ensemble d'appels de service, D est un ensemble de données et $E \subseteq (S \cup D) \times (S \cup D)$ est un ensemble de liens de dépendance entre les services exécutés et/ou les données générées et utilisées pendant l'exécution d'un workflow.*

Une notion de temps t est associée à chaque appel de service S.

Nous donnerons des définitions plus spécifiques concernant les graphes de provenance plus tard dans le texte.

Les graphes de provenances sont dirigés et acycliques [40] : chaque exécution de service ou création de donnée est liée à une estampille de temps, et le graphe ne permet aux nœuds que de relier des informations d'un temps t à des informations à $t' > t$.

Exemple 6

Considérons l'exemple de la chaine de traitements suivante : extraction de langue - traduction - extraction des entités, appliquée sur le texte de Verlaine (exemple 1 page 11). Un graphe de provenance de ce type d'exécution peut être représenté comme sur la figure 2.4.

Chaque boite bleue (contenu écrit en lettres capitales) de la figure représente l'appel à un service, tandis que les autres boites correspondent aux données sources ou dérivées. Certaines données liées aux services, comme le catalogue de lieux défini

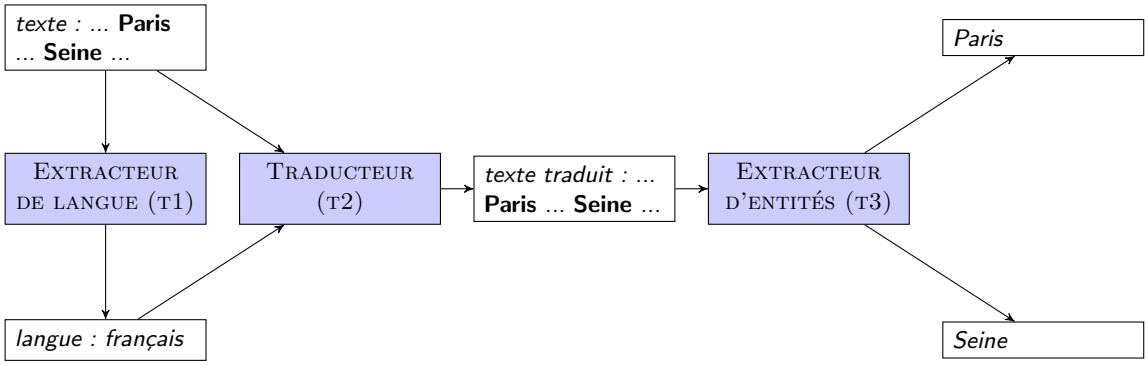


Figure 2.4 – Graphe de provenance

précédemment, n'apparaissent pas car elles sont considérées comme partie intégrante du service (si une erreur devait apparaitre dans le catalogue de lieux intégré dans l'extracteur de lieux, alors l'ensemble de l'extracteur serait impacté).

Cette définition est très générique et certaines informations manquent, telles que le type de provenance de chaque lien, ou encore le niveau de détails du graphe.

2.3.2 Utilisation de Provenance

Visualisation

La visualisation d'un graphe de provenance se fait généralement sous la forme de graphes dirigés acycliques (DAG). L'idée est de pouvoir analyser l'utilisation et la génération des données dans un workflow.

Il existe cependant des techniques de visualisation avancées qui permettent de voir les dépendances à différents niveaux de granularité [22] (ou zoom). L'idée ici est de regrouper une sous-partie d'un graphe (cluster) afin d'avoir une vision globale plus simple, et de permettre à l'utilisateur de *zoomer* sur une sous-partie pour en voir les détails.

Sur la figure 2.5, nous pouvons voir un exemple de clustering d'un graphe de provenance, dans lequel apparaissent en rouge les boites *Box1* et *Box2* représentant chacun un groupe d'exécution de services en parallèle.

Interrogation

Afin d'utiliser les informations de provenance, il faut pouvoir l'interroger de la meilleure manière. Chaque système utilise le langage de requête qui lui est le plus adapté pour stocker la provenance : SQL pour une base de données, SPARQL pour un triplestore, Xquery pour des documents XML, etc.

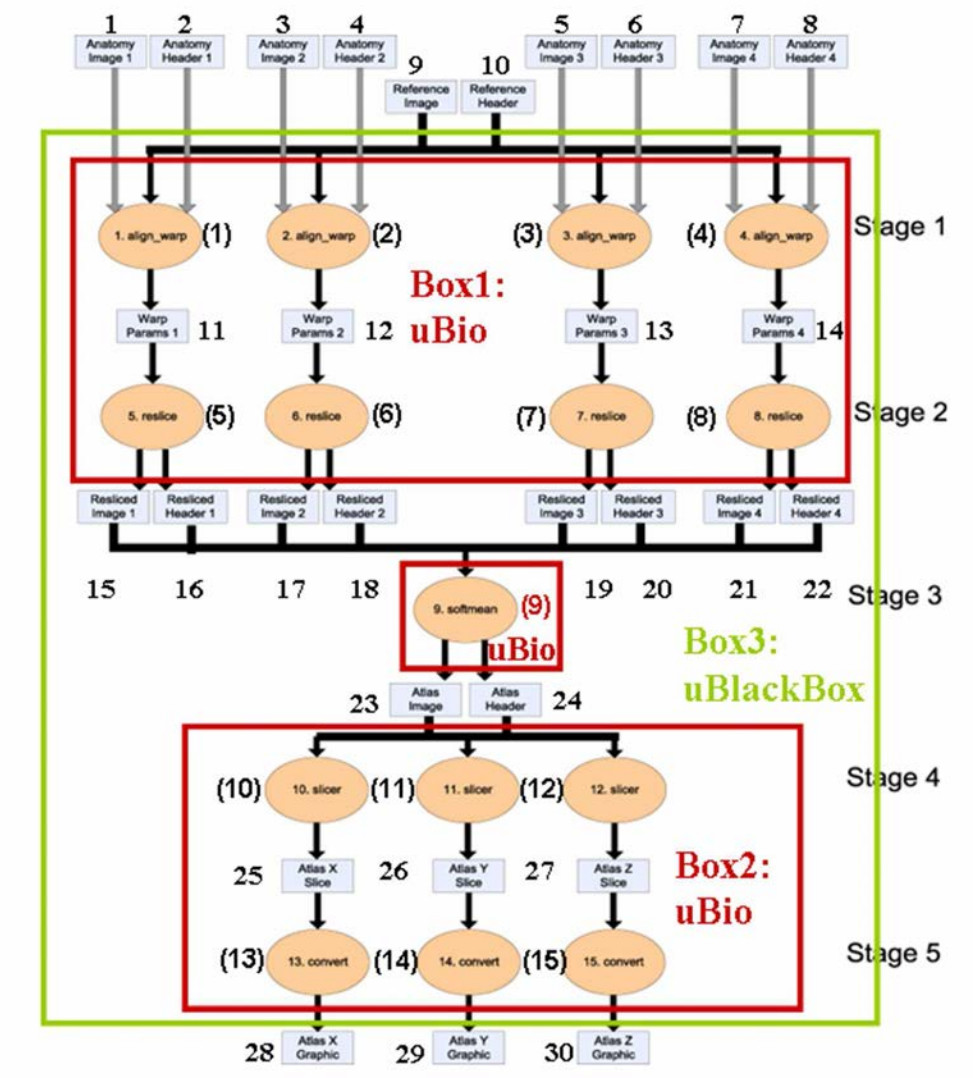


FIGURE 2.5 – Vue des clusters au sein d'un graphe de provenance

Cependant, tous ces langages ne sont pas adaptés à l'interrogation de graphes, en particulier pour formuler des expressions de chemins, où SPARQL propose des solutions plus adaptées aux graphes. Dans [6], il est proposé un langage permettant d'identifier les exécutions de workflow ayant utilisé une séquence particulière de services ou des données particulières.

Le langage présenté possède comme principaux mots-clefs *derived* et *through*. Le mot-clef *derived* permet de filtrer par rapport à une donnée, tandis que le mot-clef *through* permet de filtrer par rapport à un service.

Exemple 7

Voici quelques exemples d'expressions dans ce langage.

1. ** derived* d_1
2. d_1 *derived **
3. d_1 *derived* d_2
4. d_1 *through* s_1 *derived* d_2

La requête 1 renvoie tous les graphes de provenance ayant créé la donnée d_1*. La requête 2 renvoie tous les graphes de provenance dérivés de la donnée* d_1*. La requête 3 renvoie tous les graphes de provenance contenant un chemin entre la donnée* d_1 *et la donnée* d_2*. La requête 4 renvoie tous les graphes de provenance contenant un chemin entre la donnée* d_1 *et la donnée* d_2 *passant par le service* s_1*.*

Le langage propose également des fonctions utilisables dans les requêtes telles que *exists*, *input*, *output*, etc.

2.3.3 Standards

Afin de développer des outils génériques, des efforts de normalisation ont vu le jour sous la forme deux standards : OPM et PROV. Ces deux standards existent sous différentes formes (OWL, XML-Schema), et ont vu le jour dans le cadre d'une collaboration entre plusieurs grands noms de la provenance. Le premier, plus ancien (sorti en décembre 2007), fut adopté par un grand nombre d'outils. Le second, plus récent (devenu une recommandation en avril 2013), fut créé sous la supervision du W3C [1] avec des objectifs similaires.

Bien que différents, il est possible de représenter la majorité des graphes de provenance en utilisant l'un ou l'autre standards.

Exemple 8

Considérons une exécution de service example :service transformant une donnée example :fromValue en une donnée example :toValue. Voici les deux représentations RDF aux formats OPM et PROV de cette exécution.

1. World Wide Web Consortium

OPM :

```
example:fromValue a opmo:Artifact .
example:toValue a opmo:Artifact .

example:service a opmo:Process .

example:used a opmo:Used ;
             opmo:effect example:service ;
             opmo:cause example:fromValue .

example:wasgeneratedby a opmo:WasGeneratedBy ;
                       opmo:effect example:service ;
                       opmo:cause example:toValue .


example:wasderivedfrom a opmo:WasDerivedFrom ;
                       opmo:cause example:fromValue ;
                       opmo:effect example:toValue .
```

PROV :

```
example:fromValue a prov:Entity .
example:toValue a prov:Entity .

example:service a prov:Activity .

example:service prov:Used example:fromValue .
example:toValue prov:WasGeneratedBy example:service .
example:toValue prov:WasDerivedFrom example:fromValue .
```

On peut voir que ces deux standards sont similaires, utilisant des noms de classes identiques. La différence principale réside dans la manière de gérer les liens Used, WasGeneratedBy et WasDerivedFrom, sous la forme de classe pour OPM, et sous la forme de prédicat pour PROV.

2.3.4 Systèmes de workflows et provenance

Des outils de gestion de workflows ont vu le jour au début des années 2000, dont *Taverna* [35] et *Kepler* [4], ainsi que plus récemment, *Vis-Trails* [17]. Ces systèmes sont utilisés pour construire et exécuter des workflows scientifiques et permettent, nativement ou par le biais d'extensions, d'observer la provenance des exécutions de workflows en leur sein.

Taverna [35] est un système permettant de créer et d'exécuter des workflows. Initialement conçu pour la bio-informatique [46], puis utilisé dans d'autres domaines tels que l'astronomie [34] ou la musique [41], il permet d'exécuter des web services *SOAP* ou *REST*. *Taverna* propose également une gestion de la provenance afin d'enregistrer et d'observer les informations de provenance. L'enregistrement se fait par le biais du standard *PROV* et permet une interrogation à l'aide de requête *SPARQL*. La figure 2.6 montre l'IHM [2] de *Taverna*, sur laquelle nous pouvons voir le workflow (dans la partie droite), les services disponibles et les propriétés d'exécution.

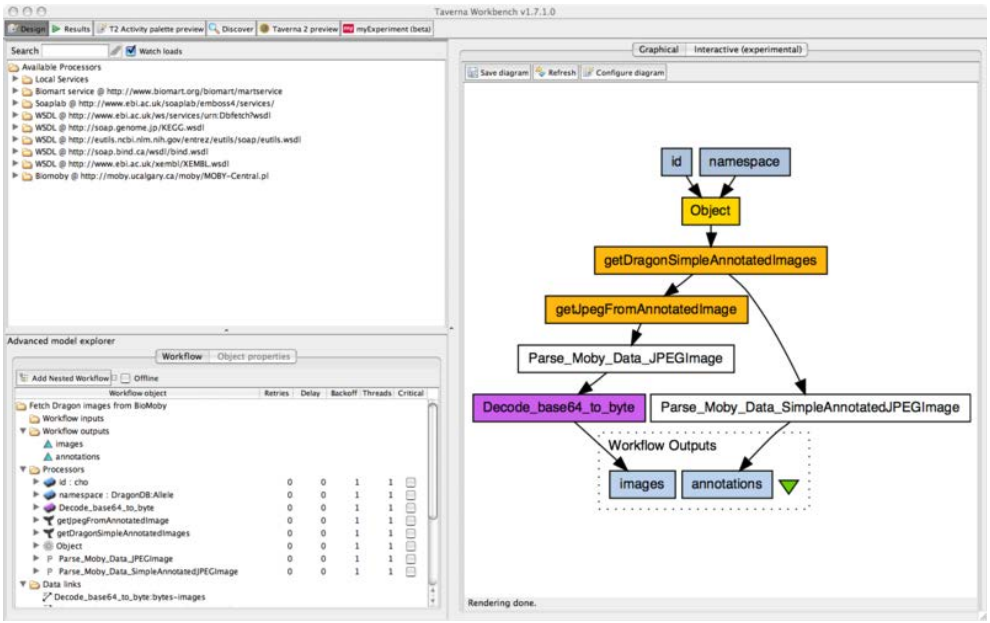


FIGURE 2.6 – IHM Taverna (https ://en.wikipedia.org/wiki/Apache_Taverna)

Kepler [4] est un système permettant de créer, d'exécuter et de partager des workflows scientifiques. La représentation des workflows se fait sous forme de graphes dirigés acycliques (DAG), où les nœuds représentent les composants à exécuter, et les arêtes sont les chemins d'exécution. L'IHM de *Kepler* permet à l'utilisateur de créer

2. Interface Homme-Machine

et exécuter des workflows, mais également de distribuer l'exécution sur plusieurs machines. *Kepler* propose également un module de provenance permettant d'enregistrer les informations de provenance, mais ne permet pas d'observer nativement ces informations. La figure 2.7 montre un workflow et un tableau regroupant différentes métadonnées d'exécution, telles que la date d'exécution et la durée.

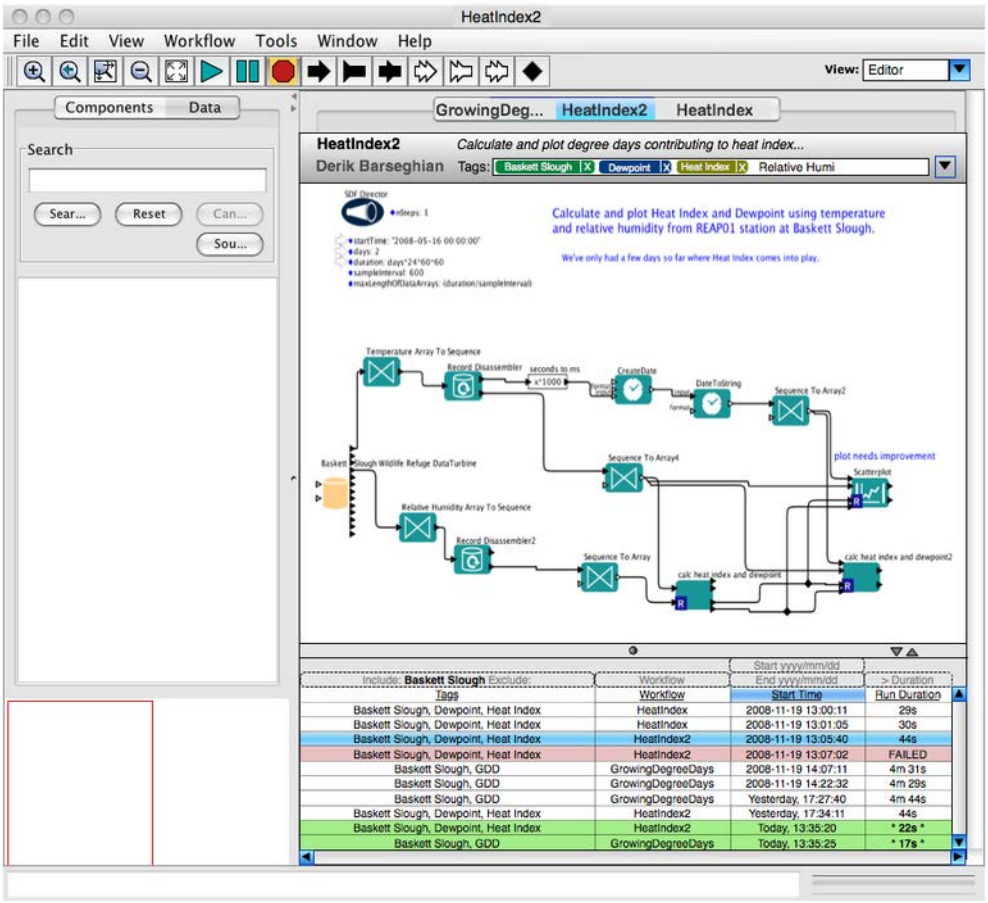


FIGURE 2.7 – IHM Kepler (https ://kepler-project.org/developers/interest-groups/provenance-interest-group/interfaces-to-provenance-for-reap)

Vis-Trails [17] est un système proposant une gestion de la provenance pour des tâches de calcul exploratoires. Il permet d'exécuter des composants faiblement couplés (type services), et offre une visualisation native des informations de provenance. *Vis-Trails* permet à l'utilisateur de faire évoluer les tâches de calcul de manière itérative afin d'explorer de nouvelles hypothèses. La figure 2.8 montre l'évolution d'un workflow après différentes modifications. Le graphe de droite montre ainsi, en couleur, les services ajoutés lors de la dernière modification.

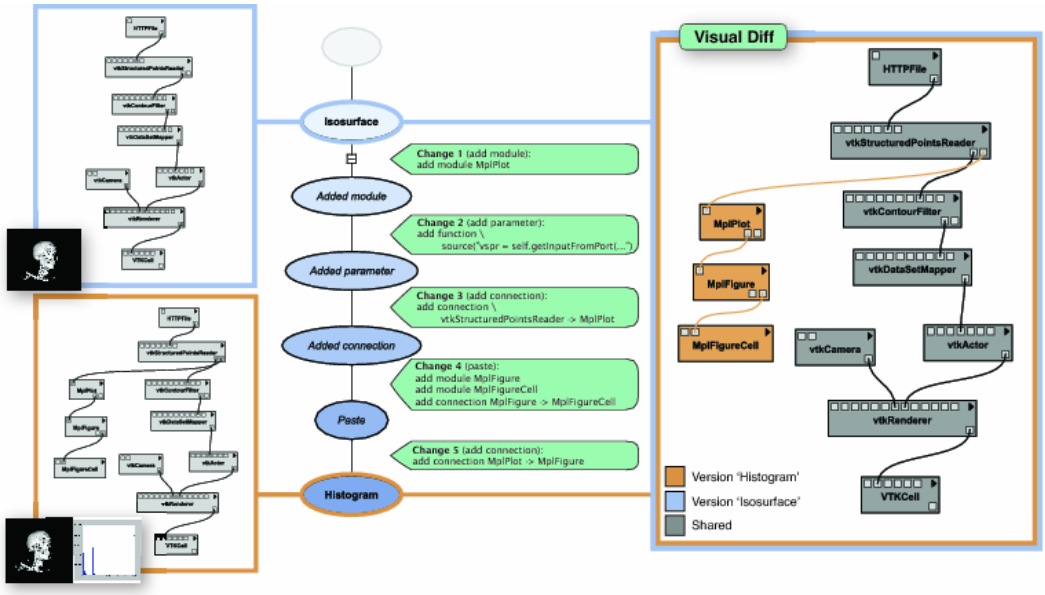


FIGURE 2.8 – IHM Vis-Trails (http ://www.aosabook.org/en/vistrails.html)

2.3.5 Provenance pourquoi/comment/où

Au sein d'un graphe de provenance, [20] et [14] distinguent trois types de lien de provenance entre les données et les services : *Why provenance*, *Where provenance*, et *How provenance*.

Exemple 9

Nous considérons pour cette partie l'exécution d'un extracteur d'entités nommées sur le texte de Verlaine. Les informations extraites par le composant peuvent être classées sous forme de deux tableaux (figure 2.9).

Le premier tableau est fourni par l'extracteur d'entités nommées, et est un extrait des lieux connus par le service. Chaque entrée donne diverses informations telles que le pays dans lequel se trouve le lieu, sa population et sa langue natale.

Le deuxième tableau contient des références aux entités géographiques du catalogue, et leur position dans le texte de Verlaine. Paris est par exemple positionnée à la ***L****igne* ***1****,* ***C****aractère* ***1****.*

La provenance **Why** correspond aux *témoins* d'une transformation. Les témoins correspondent à un sous-ensemble suffisant des données d'une exécution permettant d'assurer qu'un élément est effectivement la donnée de sortie d'un traitement. D'une manière plus concrète, cela correspond à l'ensemble des lignes (tuples) ayant servi à la création des données de sortie.

Catalogue de lieux

	Identifiant	Nom	Pays	Population	Langue
G_1 :	SeineMaritime	Seine Maritime	France	**1 251 282**	Française
G_2 :	ParisUS	Paris	Étets-Unis	24 912	Anglaise
G_3 :	Paris	Paris	France	2 249 975	Française

Entités

	Identifiant	Entité	Position dans le texte
E_1 :	SeineMaritime	Seine	L3C3
E_2 :	Paris	Paris	L1C1
E_3 :	Paris	Paris	L5C1
E_4 :	Paris	Paris	L9C1

FIGURE 2.9 – Tableaux de données

Afin d'illustrer la provenance *Why*, supposons une requête sur les tables précédentes (figure 2.9) retournant le nom d'un lieu dans le texte et la population associée, et dont le pays est la *France* :

```
SELECT DISTINCT E.Identifiant, L.Population
FROM Entité E, Lieu L
WHERE E.Identifiant = L.Identifiant
AND L.Pays = 'France'
```

Résultat :

Identifiant	Population	
SeineMaritime	1 251 282	$\{E_1, G_1\}$
Paris	2 249 975	$\{E_2, E_3, E_4, G_3\}$

Nous pouvons voir que la requête nous retourne deux résultats. Le premier résultat, $\{$*Seine ;1 251 282*$\}$, possède comme témoins les tuples E_1 et G_1 qui sont requis et suffisants pour générer la donnée de sortie $\{$*Seine ;1 251 282*$\}$ à partir de cette requête. Il en est de même pour la deuxième donnée $\{$*Paris ;2 249 975*$\}$ et les témoins $\{E_2, E_3, E_4, G_3\}$, correspondant à toutes les lignes ayant été utiles à la création du résultat.

La provenance **Where** correspond aux éléments à la *source* d'une transformation de données. Cette *source* est la localisation d'une donnée, ainsi, quand les témoins

d'une provenance correspondent aux lignes ayant servi à la création de la sortie, les sources sont les champs d'où les données ont été copiées ou à partir desquels une transformation a eu lieu [39].

En reprenant le même exemple, la provenance *where* de l'élément *Seine* du premier résultat est (E_1, Identifiant), E_1 étant la ligne et *Identifiant* le champ duquel l'information est extraite. De la même manière, la provenance *where* de *2 249 975* est la position (G_3, Population).

Contrairement à la provenance *why*, il est possible pour plusieurs requêtes similaires (même résultats) d'avoir une provenance *where* différente. Considérons la requête ci-dessous :

```
SELECT DISTINCT L.Identifiant, L.Population
FROM Entité E, Lieu L
WHERE E.Identifiant = L.Identifiant
AND L.Pays = 'France'
```

Cette requête renvoie exactement le même résultat que la première avec la même provenance *why* pour chacune des lignes du tableau. Il y a cependant une différence dans la provenance *where* pour le résultat *Seine*. En effet, quand avec la première requête nous obtenions la position (E_1, Identifiant), nous obtenons maintenant (G_1, Identifiant). Il en va de même pour *Paris*, passant de (E_2, E_3, E_4, Identifiant) à (G_3, Identifiant).

La provenance **How** *explique* la transformation des données elles-mêmes. On peut décrire ce type de provenance comme le chemin parcouru par les données pour créer les données de sortie. Green et al. [31] définit ce type de provenance comme étant modélisable sous la forme de polynômes où . est une jointure entre deux tuples, et + est une union.

Les résultats de la requête précédente possèdent cette provenance *how* : le couple *Seine ;1 251 282* est issu de la transformation $E_1.G_1$ (c'est-à-dire la combinaison des deux tuples E_1 et G_1) et *Paris ;2 249 975* est issu de $E_2.G_3$. La provenance *how* prend tout son sens pour les requêtes plus complexes. Prenons par exemple la requête suivante qui retourne les positions d'un lieu dans le texte existant dans plusieurs pays, ainsi que le nombre de pays dans lesquels il apparait :

```
SELECT S.Position, COUNT(L1.Pays)
FROM Lieu L1,
(SELECT E.Position, L2.Nom
FROM Entité E, Lieu L2
WHERE E.Identifiant = L2.Identifiant) S
WHERE S.Nom = L1.Nom
```

Résultat :

S.Position	COUNT(L1.Pays)	
L3C3	1	$E_1.G_1$
L1C1	2	$E_2.G_3.G_2$
L5C1	2	$E_3.G_3.G_2$
L9C1	2	$E_4.G_3.G_2$

Les résultats obtenus correspondent aux lignes de la base de données jointes entre elles. Ainsi le résultat de la ligne 2 est obtenu en effectuant une jointure entre E_2 et G_3 au sein du *SELECT* imbriqué (*WHERE E.Identifiant = L2.Identifiant*). La jointure avec G_2 se fait dans la clause *WHERE* entre le résultat du *SELECT* imbriqué et le tableau de lieux (*WHERE S.Nom = L1.Nom*).

Malgré une requête plus complexe, nous pouvons voir que la provenance *how* est restée relativement simple.

Dans le cas des workflows, la provenance *how* décrit la transformation exercée par un composant. Les polynômes permettraient dans ce contexte d'identifier les données utilisées par un service, ainsi que les données créées. Dans le cadre de la plateforme WebLab, cela peut se faire en identifiant les nœuds XML en entrée et en sortie de l'exécution d'un service, afin de générer un graphe de provenance entre les informations.

2.3.6 Granularité de la Provenance

La provenance des informations peut être plus ou moins précise suivant les besoins et les possibilités. On parle de provenance grains-fins (*fine-grained*) et de provenance grains-grossiers (*coarse-grained*).

La granularité de la provenance correspond au niveau de détail avec lequel les liens sont représentés. Cette granularité est définie ainsi :

$$Grains\ Grossiers(p) \subseteq Granularité(p) \subseteq Grains\ Fins(p)$$

avec p la provenance représentée, et les fonctions *Grains Grossiers* et *Grains Fins*, respectivement les fonctions la moins et la plus détaillées.

Cette inclusion ($\subseteq$) signifie que l'ensemble des informations de provenance contenues dans le graphe *Grains Grossiers(p)* sont présentes au sein du graphe *Grains Fins(p)*.

La granularité d'un lien de provenance peut donc potentiellement se définir sur de nombreux niveaux, et il est envisageable de *zoomer/dézoomer* sur un graphe de provenance pour obtenir plus ou moins d'informations suivant les besoins [25].

Dans le contexte des workflows, la granularité est représentée par la connaissance du fonctionnement des services. En effet, souvent, les workflows exécutent des services dont le fonctionnement intrinsèque n'est pas connu (services *black-box*), à l'opposé des services *white-box* dont le fonctionnement est connu.

Un service *black-box* crée des liens de provenance entre l'ensemble des données d'entrée et l'ensemble des données de sortie et a donc une provenance grains grossiers, tandis qu'un service *white-box* crée des liens de provenance plus fins entre chaque information, et a donc une provenance grains fins. Une troisième notion, celle de *grey-box*, correspond à une granularité intermédiaire.

Certains modèles [5] capturent l'état interne de chaque composant et l'exposent afin d'éclaircir son fonctionnement, passant ainsi de *black-box* à *grey-box* ou *white-box*. L'avantage de ce type de modèle, outre une provenance plus fine, est la possibilité de *zoom* en passant d'une provenance au niveau *black-box* à un niveau plus détaillé.

FIGURE 2.10 – Service black-box

Exemple 10

Prenons l'exemple des figures 2.10, 2.11 et 2.12. La figure 2.10 montre un service black-box de traduction en anglais. Nous ne connaissons rien de son fonctionnement hormis qu'il prend en entrée un texte, et qu'il produit la traduction de ce texte en sortie. À l'opposé, la figure 2.12 (service white-box) montre le fonctionnement présent du traducteur : un premier composant va compter la fréquence d'apparition de chaque

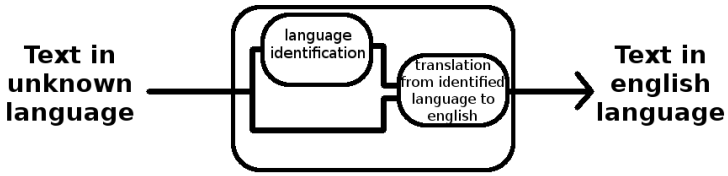


FIGURE 2.11 – Service grey-box

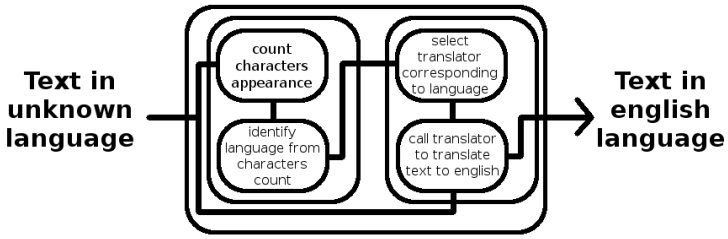


FIGURE 2.12 – Service white-box

caractère, puis une autre partie va en déduire la langue du texte en entrée, permettant de choisir le traducteur correspondant pour traduire le texte. La figure 2.11 montre un service dit grey-box, situé entre les deux : le traducteur identifie la langue du texte puis le traduit.

2.3.7 Génération de la provenance

Nous verrons dans cette section les méthodes de génération de la provenance, ainsi que les implémentations déjà existantes.

On peut distinguer deux méthodes pour la génération de la provenance : la méthode prospective et la méthode rétrospective.

Définition 3 *Provenance prospective [27]*

La provenance prospective est fondée sur une spécification d'un workflow abstrait comme un guide permettant de déduire les informations de provenance. Cette spécification est indépendante de toute exécution, et ne contient que les informations nécessaires suffisantes pour créer un graphe de provenance.

La provenance prospective demande une connaissance parfaite des transformations des données afin de recréer à l'identique les résultats obtenus, ce qui la rend peu commune, surtout lorsque l'on considère les services exécutés comme black-box.

Exemple 11

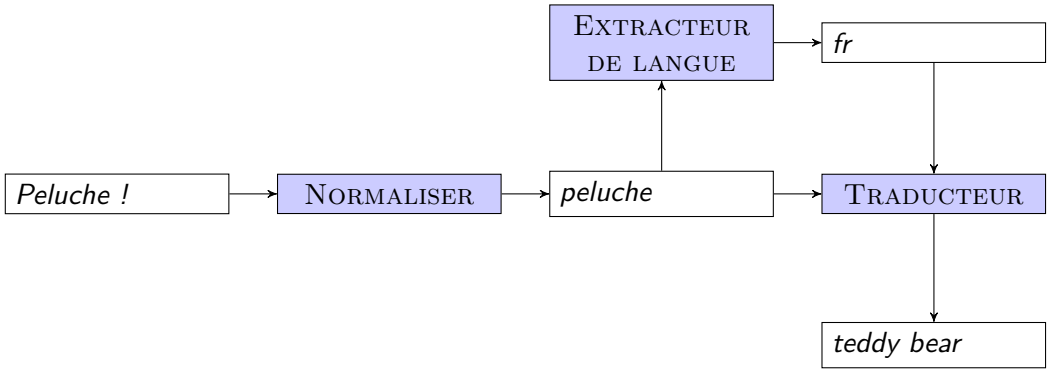


Figure 2.13 – Graphe de provenance : Peluche

Nous considérons comme exemple l'exécution d'une chaine de traitements media-mining, transformant une chaine de caractères (ici, "Peluche !") au travers d'un normaliseur ("peluche"), d'un extracteur de langue ("fr") et d'un traducteur anglais ("teddy bear"). Le graphe de provenance de cette exécution est visible sur la figure 2.13.

Donnée d'entrée	*Peluche !*
Exécution 1	Normaliser
Exécution 2	Extracteur de langue
Exécution 3	Traducteur

Figure 2.14 – Provenance prospective : exemple

Le tableau de la figure 2.14 montre une spécification de provenance prospective pour notre exemple de workflow. Les données enregistrées sont : la donnée d'entrée, puis les services exécutés dans l'ordre. L'idée ici est que la transformation de données orchestrée par les services est connue (service white-box), permettant de déduire un graphe de provenance détaillé à partir de la donnée d'entrée.

Une alternative à la provenance prospective est la gestion par *inversion* [44]. Quand la provenance prospective recrée les informations intermédiaires et finales à partir des données d'entrées et de fonction de transformation, la provenance par inversion recrée les données d'entrées et intermédiaires par l'application de fonctions de transformation inverse appliquées sur les données de sortie.

Exemple 12

Le tableau ci-dessus 2.15 montre une spécification de provenance par inversion pour notre exemple. Les données enregistrées sont la donnée de sortie et les services

Donnée de sortie	*teddy bear*
Exécution 1	Normaliser
Exécution 2	Extracteur de langue
Exécution 3	Traducteur

FIGURE 2.15 – Provenance par inversion : exemple

exécutés dans l'ordre. L'idée ici, de la même manière que pour la provenance prospective, est que la transformation de données orchestrée par les services est connue (service white-box), permettant de déduire un graphe de provenance détaillé à partir de la donnée de sortie. D'un point de vue pratique, il est difficile dans le cas actuel de déduire la langue en entrée ("fr") à partir de la traduction ("teddy bear"), ou encore de restituer la ponctuation ("Peluche !") à partir de la version normalisée ("peluche"). Cette méthode n'est donc utilisable que dans les cas où l'inversion est possible.

Définition 4 *Provenance rétrospective [27]*

La méthode rétrospective utilise des informations produites par les exécutions passées d'un workflow. Elle consiste à enregistrer à chaque étape de l'exécution toutes les informations nécessaires (composants exécutés, données d'entrées, de sorties, temps de traitement, etc.), appelées traces d'exécution, afin de pouvoir créer un graphe de provenance.

La méthode rétrospective est souvent plus facile à mettre en place, mais demande davantage d'espace de stockage. Cette méthode est populaire car elle ne nécessite pas de connaitre le fonctionnement des composants ayant été exécutés, et permet de raisonner plus facilement sans avoir à recalculer chaque donnée intermédiaire.

Exemple 13

Entrée Normaliser	*Peluche !*
Sortie Normaliser	*peluche*
Entrée Extracteur de langue	*peluche*
Sortie Extracteur de langue	*fr*
Entrée Traducteur	*peluche, fr*
Sortie Traducteur	*teddy bear*

FIGURE 2.16 – Provenance rétrospective : exemple

Le tableau 2.16 montre les enregistrements d'une provenance rétrospective. Les données intermédiaires sont explicitement renseignées. On peut voir que l'exécution du Normaliser a transformé la valeur "Peluche !" en "peluche", puis l'Extracteur

de langue utilise cette nouvelle valeur pour identifier la langue française. Enfin le traducteur utilise les deux valeurs "peluche" et "fr" pour obtenir la donnée finale "teddy bear". Aucune autre information que celles présentes dans le tableau n'est nécessaire pour créer le graphe de provenance des données.

Nous pouvons voir que cette méthode demande plus d'espace de stockage. En effet, ici, elle demande 6 lignes de tableau, alors que les méthodes précédentes n'en ont demandé que 4, d'où une augmentation de 50%.

2.4 Conclusion

Dans ce chapitre, nous avons abordé et défini les notions liées à notre problématique de qualité dans une orchestration de services WebLab. Nous avons défini la qualité d'une information comme un ensemble de dimensions dépendant du contexte, présenté les différentes méthodes de modélisation et de génération de graphes de provenance, ainsi que les standards liés à leur représentation.

Nous présentons dans les chapitres suivants le modèle de provenance que nous avons créé pour le système WebLab (chapitre 3), puis le modèle de qualité que nous avons rattaché à celui-ci (chapitre 4). Nous finirons par présenter une implémentation (chapitre 5) de notre solution au sein d'un prototype proche des problèmes à résoudre dans les projets WebLab.

3

Modèle de provenance

Nous proposons dans ce chapitre un modèle de génération de graphe de provenance au sein de la plateforme WebLab, ou de tout autre système fondé sur un stockage en documents XML.

Dans la première section, nous introduisons le fonctionnement de la plateforme WebLab pour la gestion des données sous forme de documents XML.

Dans la deuxième section, nous présentons notre première contribution : un modèle de provenance pour WebLab. Nous commençons par définir ce que nous appelons un graphe de provenance dans le contexte de la plateforme, puis nous présentons nos méthodes d'extraction et de stockage de ce graphe.

Dans les troisième et quatrième sections, nous présentons des améliorations possibles à notre système.

3.1 Documents WebLab

L'exécution d'un workflow WebLab utilise des documents XML pour faire transiter les informations de service en service. Chaque média (image, vidéo, texte) inclus dans le document possède son propre nœud (AudioMediaUnit, TextMediaUnit) au sein du document XML. Chacun de ces nœuds peut posséder un ou plusieurs sous-nœuds pouvant renfermer des informations telles que le contenu du média, la langue d'un texte, ou la résolution d'une image. La figure 3.1 ci-dessous montre un exemple de document, extrait du traitement présenté dans l'introduction.

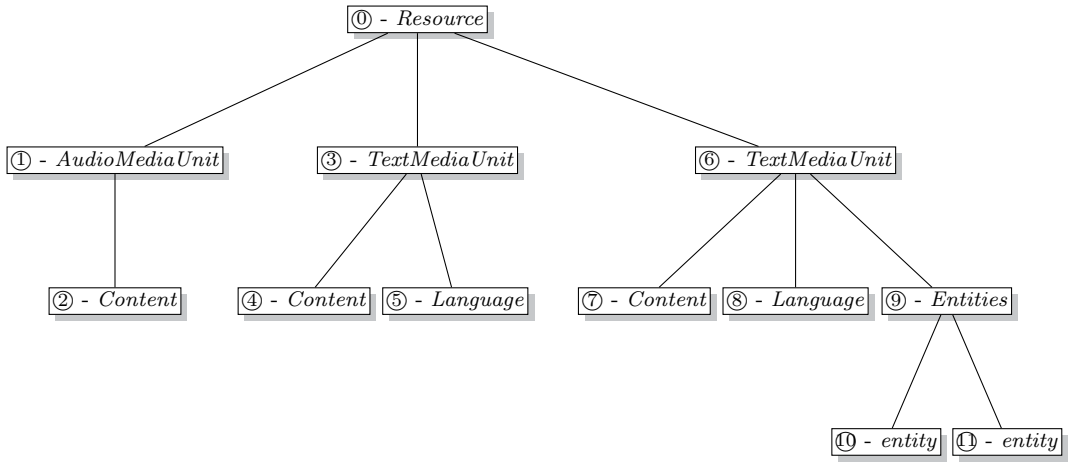


FIGURE 3.1 – Exemple de document WebLab

Certains nœuds intermédiaires, non identifiés, ne sont pas représentés ici pour faciliter la compréhension. Chacun des nœuds de la figure est identifié à l'aide d'une URI [1], et correspond aux *ressources* utilisables par les services.

Définition 5 *Document WebLab*

Un document WebLab d se définit comme couple (θ, r), où θ est un arbre XML, et r est un ensemble de ressources identifié à l'aide d'une URI.

Chaque ressource r représente une donnée utilisable par les services, et éventuellement créée par l'un d'entre eux.

Chaque appel de service utilise un ou plusieurs ensembles de ressources afin de créer de nouvelles ressources. Ainsi, un extracteur de langue utilisera une ressource de type *Content* (④) de manière à créer une nouvelle annotation de type *Language* (⑤).

Une spécificité des documents WebLab est l'impossibilité de supprimer un nœud dans le document. En effet, chaque service exécuté au sein de la plateforme ajoute de nouvelles informations à l'arbre XML reçu en entrée afin de l'enrichir. Cette sémantique d'"ajout" d'information garantit une augmentation constante de l'information contenue dans les documents WebLab. Le résultat de l'exécution d'un workflow est donc un document XML contenant l'ensemble des informations utilisées et produites par les services présents dans le workflow. Nous supposons que les paramètres d'appels des services sont contenus au sein de ce document, et peuvent être extraits par l'orchestrateur de services avant l'appel d'un service. Le modèle WebLab est ainsi compatible avec la notion de *"nested data collections"* [7] (collections de données imbriquées).

Lors de l'exécution d'une orchestration de services, le document WebLab est modifié par chaque appel de service, et évolue donc avec le temps. De manière à représenter cette évolution, nous annotons les ressources à l'aide d'estampilles de temps.

Nous pouvons voir sur la figure 3.2 une nouvelle représentation du document de la précédente figure 3.1. Sur ce document, nous pouvons voir les services ayant créé les nœuds ainsi que l'estampille de temps correspondant. Nous pouvons voir par exemple que le traducteur (*translator*) a créé un nœud *TextMediaUnit* au temps $t4$.

Notre exemple contient ainsi l'ensemble des informations issues de l'exécution séquentielle des opérations : transcription, extraction de la langue, traduction, et

1. Uniform Resource Identifier

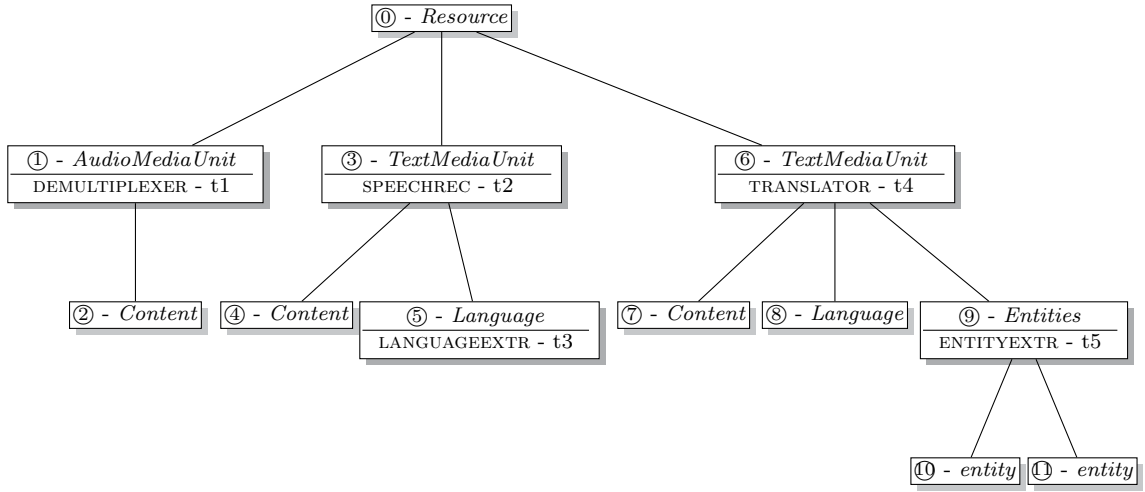


FIGURE 3.2 – Exemple de document WebLab avec annotation temporelle

l'extraction d'entités nommées. Est également présent sur le document le nœud AudioMediaUnit issu du démultiplexeur, qui sera utilisé en entrée du service de transcription.

Chaque service WebLab utilise un sous-ensemble de ressources identifiées à l'aide d'un ou plusieurs URI. À chaque ressource correspond un sous-arbre du document XML incluant toutes les ressources et tous les sous-nœuds non-identifiés antérieurs à l'exécution du service. De la même manière, la sortie d'un service est identifiée par un ou plusieurs URI et l'ensemble de leurs sous-nœuds créés par l'exécution du service.

Dans l'exemple du traducteur, liant la TextMediaUnit (3) à la TextMediaUnit (6), l'ensemble des sous-nœuds des deux MediaUnit sont inclus dans la relation à l'exception des nœuds *entities* et *entity* car ayant étant créés à posteriori de l'exécution du traducteur.

3.2 Modèle de provenance WebLab

Notre première contribution est la définition de ce qu'est un graphe de provenance dans le contexte de la plateforme WebLab, ainsi qu'une méthode de génération à partir des traces d'exécution.

3.2.1 Graphe de provenance WebLab

Nous avons défini dans le chapitre précédent (chapitre 2) qu'un graphe de provenance est une représentation des liens de dépendance entre les services exécutés et/ou les données générées et utilisées pendant l'exécution d'une chaine de traitements.

La création d'un graphe de provenance à partir de l'exécution d'un service se fera par la mise en relation de l'ensemble des données de sortie du service avec l'ensemble des données d'entrée. Mis bout-à-bout dans une orchestration de services, il en ressort un graphe de provenance permettant de connaitre toutes les données et les transformations à l'origine d'une information.

Exemple 14

En prenant l'exemple de la figure 3.1 et l'exécution des services associés, le graphe de provenance recherché correspond à celui de la figure 3.3.

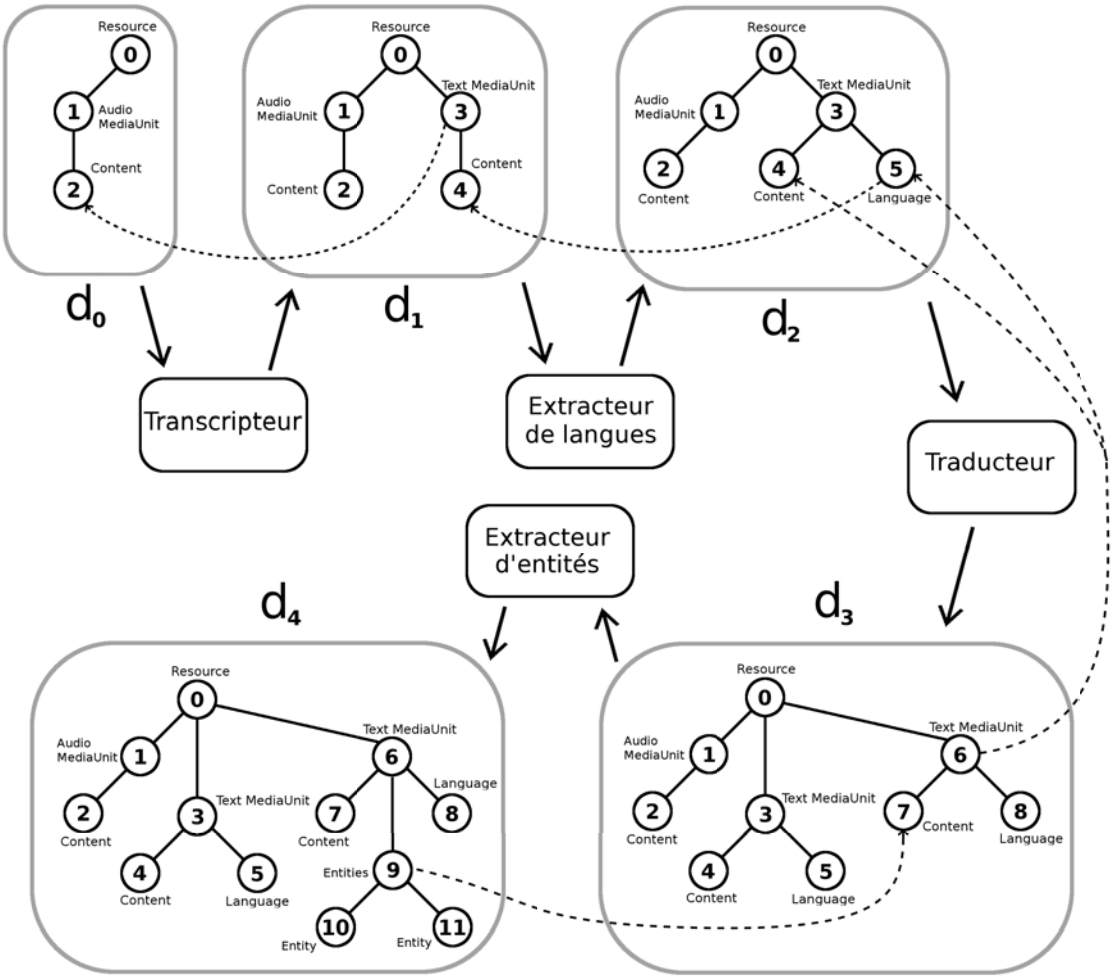


FIGURE 3.3 – Graphe de provenance

Sur le graphe, nous pouvons voir l'évolution du document XML au fur et à mesure du temps avec l'exécution de chaque service. Les flèches pointillées visibles entre les documents montrent les liens de provenance entre les données.

Ainsi, le graphe nous montre que le nœud ② Content est utilisé par le service de Transcription pour créer le nœud ③ TextMediaUnit et son fils ④ Content. Ce dernier permet à l'Extracteur de langue de créer le nœud ⑤ Language, et ces deux derniers sont utilisés par le Traducteur pour créer un nouveau nœud ⑥ TextMediaUnit et ses deux fils, ⑦ Content et ⑧ Language. Le fils ⑦ Content est finalement utilisé pour créer les derniers nœuds, ⑨ Entities et ses deux fils par l'Extracteur d'entités nommées.

Le graphe de provenance montré dans la figure 3.3 correspond au graphe que l'on souhaiterait obtenir à partir de l'ensemble des données d'une exécution.

Notre idée est, partant d'un graphe de provenance dit naïf, de raffiner de plus en plus le graphe de provenance en substituant les liens superflus afin de se rapprocher ou d'atteindre un graphe dit parfait :

$$Graphe_{parfait}(d) \subseteq Graphe(d) \subseteq Graphe_{naif}(d)$$

Nous appelons *graphe naïf* un graphe de provenance dont l'ensemble des ressources XML sont reliées entre elles, jusqu'à atteindre les limites structurelles du document XML : un nœud parent ne peut pas dépendre d'un de ses enfants. Nous appelons *graphe parfait* le graphe de provenance contenant les liens réels entre les données.

Il est possible qu'un graphe ne respecte pas l'inclusion de graphe précédente dans le cas d'un ou plusieurs liens de provenance manquants. Nous considérons dans ce cas que le graphe est invalide et qu'une erreur a été commise dans le traitement de la provenance.

Nous allons dans les sections suivantes raffiner le graphe de provenance. Nous allons commencer par utiliser les règles structurelles de XML et des règles définies sur chaque service afin d'identifier les liens de provenance potentiels qu'un service a pu créer. Nous nommons ce type de graphe de structurel.

Nous continuerons avec l'utilisation des estampilles temporelles associées aux ressources lors de l'exécution des services afin de respecter les contraintes temporelles de l'exécution séquentielle de services. Ces contraintes étant imposées après obtention du graphe structurel, les graphes respectent l'inclusion $Graphe_{temporel}(d) \subseteq Graphe_{structurel}(d)$.

Nous aurons au final pour chaque étape le respect de l'inclusion de graphe :

$$Graphe_{parfait}(d) \subseteq Graphe_{temporel}(d) \subseteq Graphe_{structurel}(d) \subseteq Graphe_{naif}(d)$$

3.2.2 Génération du graphe structurel

Une difficulté particulière pour la génération d'un graphe de provenance dans la plateforme WebLab est l'absence d'information sur le fonctionnement des services. En effet, WebLab permet une intégration de composants tiers dont le fonctionnement est inconnu (services black-box). La solution que nous avons adoptée exploite une autre propriété importante de la plateforme qui ne permet aux services que d'ajouter des nouveaux fragments XML (sans pouvoir effacer ou modifier les fragments existants). Grâce à cette restriction, il est possible, par comparaison des documents d'entrée et de sortie, de connaitre les nouveaux nœuds générés par chaque appel de service. Afin d'inférer des informations de provenance plus précises, nous offrons la possibilité aux fournisseurs d'un service tiers de préciser la transformation faite par le service à l'aide d'expressions spécifiant la dépendance entre les données d'entrées et les données de sorties.

Nos règles de dépendance des données sont fondées sur un langage de requête XPath enrichie. Nous utilisons ici XPath Core, les possibilités offertes par celui-ci étant suffisante pour démontrer l'intérêt de notre modèle.

Définition 6 *Chemin XPath*

Un chemin XPath est une séquence d'étapes de la forme :

$$étape_1/étape_2/\ldots/étape_k$$

où chaque $étape_i$ vaut :

$$étape_i = axe :: noeud[prédicat]^*$$

où axe, nœud et prédicat sont des éléments de XPath. L'axe correspond à la direction dans laquelle on veut se déplacer dans l'arbre XML. Différents axes existent tels que child, descendant ou ancestor. Ce champ est optionnel, ayant la valeur par défaut child (/child : :filtre = /filtre), ou descendant-or-self si le slash est doublé (/descendant-or-self : :node()/filtre = //filtre). Le champ nœud correspond ensuite au nom ou au type du nœud recherché suivant l'axe utilisé, et le prédicat, écrit entre crochets, est ensuite utilisé pour filtrer les nœuds précédemment sélectionnés.

Ce type de chemin nous permet d'identifier un ensemble de ressources. Par exemple, le chemin :

$$//TextMediaUnit$$

retourne l'ensemble des nœuds TextMediaUnit du document WebLab.

Dans le but de créer des liens de provenance entre plusieurs ensembles de ressources, nous avons étendu les expressions XPath avec des variables de liaisons nous permettant de faire correspondre des ressources entre plusieurs chemins XPath. L'idée est de créer un graphe de provenance avec une granularité plus fine.

Définition 7 *Chemin XPath enrichi*

Un chemin XPath enrichi (ou patron XPath) est une séquence d'étapes de la forme :

$$étape_1/étape_2/\ldots/étape_k$$

où chaque $étape_i$ est une étape XPath :

$$étape_i = axe :: noeud[prédicat]^*[\alpha_i]^{0,1}$$

où axe, nœud et prédicat sont les éléments XPath définis précédemment, et α_i est une séquence optionnelle d'affectation de variables de liaisons du type :

$$x_i := @a_1, \ldots, x_j := @a_j, \ldots, x_n := @a_n$$

Les expressions $@a_j$ sont des attributs XML appelés attributs de liaison et $@x_j$ sont des variables de liaison permettant de faire correspondre les ressources entre plusieurs chemins XPath.

Un champ α_i peut également prendre comme valeur uniquement x_i, l'attribut enregistré dans la variable sera alors implicitement l'*URI* du nœud.

Par la suite, nous utiliserons la notation $\varphi_(\bar{x})$ où φ_x est un ensemble de variables de liens présent dans l'expression, à l'exception de la variable implicite $\$r$.

Il est possible à l'aide de ces expressions de relever des ensembles de nœuds de notre exemple précédent sur la figure 3.4.

Exemple 15

$\varphi_1 = /resource/AudioMediaUnit/Content$
$\varphi_2(\$x) = //TextMediaUnit[\$x]/Content$
$\varphi_3(\$x) = //TextMediaUnit[\$x][.//Language =' fr']$

L'expression φ_1 ne possède pas de variable d'affectation explicite, et retourne le nœud Content (2) sous le nœud AudioMediaUnit (1).

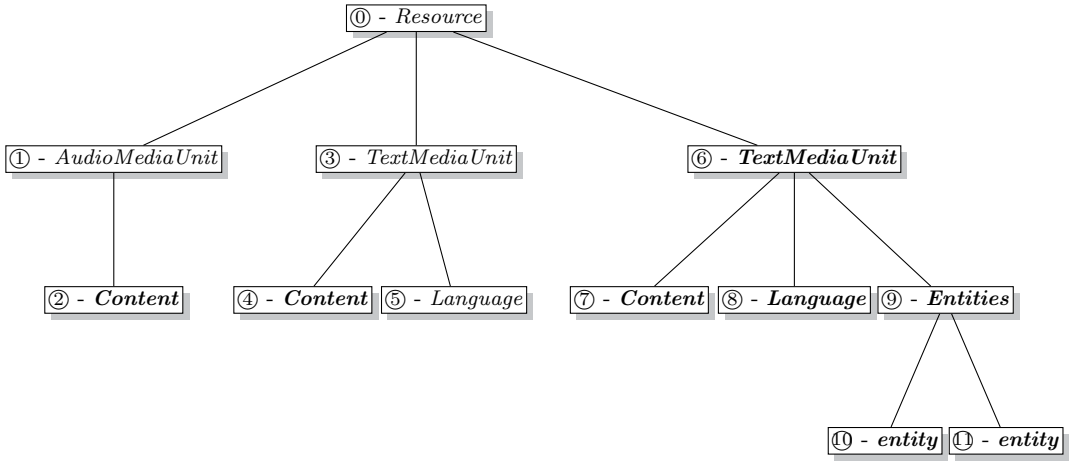


FIGURE 3.4 – Exemple de document WebLab

L'expression φ_2 retourne les sous-nœuds Content (4) et (7) des TextMediaUnit (3) et (6) tout en affectant la variable $x aux nœuds (3) et (6).

Enfin, la dernière expression φ_3 ne retourne que la TextMediaUnit dont la langue identifiée est Français. Seule la ressource (6) possède un sous-nœud Language dont la valeur est 'fr' et est donc retournée.

Les chemins XPath enrichis permettent de sélectionner un sous-ensemble de ressources dans un document et d'affecter des ressources à des variables partagées. Afin de créer des liens de provenance, nous devons relier des sous-ensembles entre eux. Nous aurons ainsi un ensemble de ressources identifiées comme source d'un autre ensemble de nœuds.

Définition 8 *Règle de dépendance des données*

Nous définissons les dépendances des données lors d'une opération comme un lien entre des nœuds Sources et des nœuds Cibles. Les nœuds Sources correspondent aux nœuds d'entrées de l'opération, et les nœuds Cibles aux nœuds de sortie de cette même opération.

Une règle de dépendance de données (ou règle de mapping) est définie comme une expression

$$\varphi_{S0}(\bar{x})[,\varphi_{Si}(\bar{x})]^* \rightarrow \varphi_{C0}(\bar{x})[,\varphi_{Ci}(\bar{x})]^*$$

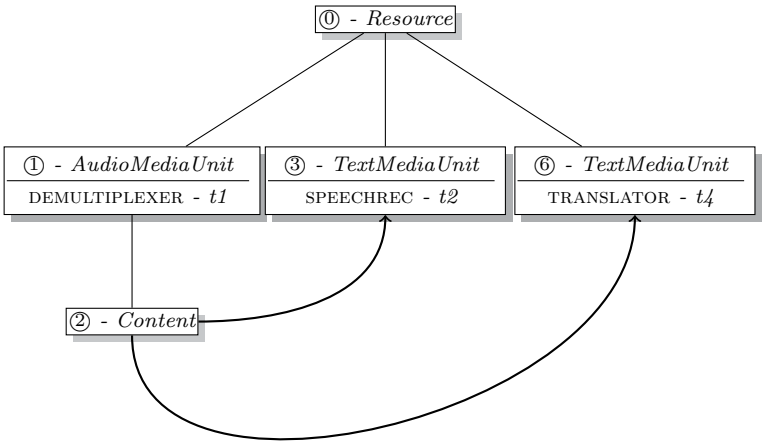
où $\varphi_S(\bar{x})$ et $\varphi_C(\bar{x})$ sont des expressions XPath vues précédemment. Les premières expressions représentent les nœuds Sources et les secondes les nœuds Cibles.

Il est possible d'avoir un ensemble d'expressions XPath séparées par des virgules afin de créer des règles de dépendances plus complexes, où deux nœuds Sources, ou plus, peuvent être utilisés pour créer un, ou plusieurs, nœuds Cibles.

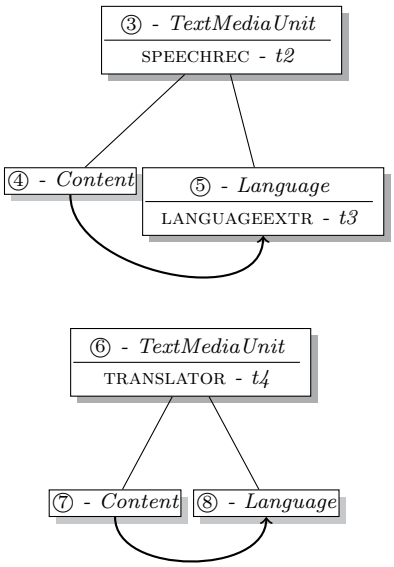
Exemple 16

Transcripteur : $/resource/AudioMediaUnit/Content \rightarrow //TextMediaUnit$

La première expression correspond à la règle de dépendance du Transcripteur, transformant le nœud Content en une nouvelle TextMediaUnit. L'application de la règle de manière naïve retourne un lien de provenance pour chaque MediaUnit présente dans le document. Nous verrons dans la suite comment filtrer les faux positifs.



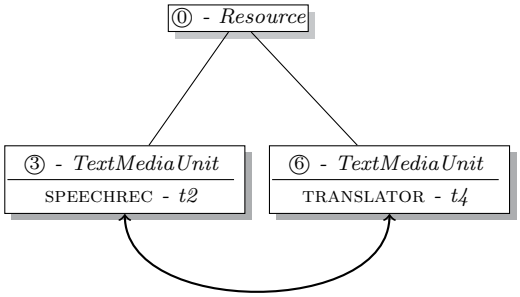
Extracteur de langue : $//TextMediaUnit[\$x]/Content \rightarrow //TextMediaUnit[\$x]/Language$



La deuxième règle décrit l'extraction de la langue, créant une annotation Language dans une TextMediaUnit à partir du Content. L'utilisation de la variable $\$x$ crée un lien de provenance entre l'annotation de langue et le contenu d'une même TextMediaUnit.

Traducteur : $//TextMediaUnit[//Language] \rightarrow //TextMediaUnit$

Enfin la dernière expression crée un lien entre une TextMediaUnit et toute autre TextMediaUnit dont la langue est identifiée.



Nous pouvons voir dans les exemples précédents des liens de provenance incohérents entre eux, tels que le dernier, proposant un lien de provenance bidirectionnel. Afin d'affiner le graphe, nous allons utiliser des variables au sein de nos règles, puis des annotations de temps.

Le recours aux variables permet diverses utilisations. Il est par exemple possible d'utiliser les variables de liaisons uniquement dans la partie source afin d'obtenir une granularité de provenance plus fine. Ainsi, la règle

$$//TextMediaUnit[\$x]/Content, //TextMediaUnit[\$x]/Language \rightarrow //TextMediaUnit$$

permet d'identifier comme source les deux sous-nœuds, *Content* et *Language*, d'une même *TextMediaUnit*, afin de créer une *TextMediaUnit*.

La règle

$$//TextMediaUnit[\$x]/Content, //TextMediaUnit[\$x]/Language \rightarrow //TextMediaUnit[@uri \neq \$x]$$

permet par exemple de lier les mêmes sous-nœuds *Content* et *Language* d'une *TextMediaUnit* commune à une *TextMediaUnit* différente de celle-ci.

3.2.3 Sémantique structurelle

L'évaluation d'une expression XPath $\varphi(\bar{x})$ sur un document WebLab consiste à trouver l'ensemble des homomorphismes de l'arbre de requête de l'expression XPath vers le document XML.

Définition 9 *Tuple de liaison*

Chaque expression XPath $p = \varphi(\bar{x})$ appliquée à un document XML (WebLab) d définit un ensemble d'homomorphismes (fonctions de liaison) $h_i : \bar{x} \rightarrow r$ des variables $\bar{x}$ vers l'ensemble des ressources r dans d tel que $\varphi(h_i(\bar{x}))$ respecte toutes les

contraintes structurelles présentes dans le document d. Le tuple $h_i(\bar{x})$ est appelé un tuple de liaison.

À chaque tuple de liaison obtenu précédemment correspond un sous-arbre XML du document WebLab répondant au patron XPath.

Définition 10 *Sémantique XPath*

Le résultat $\mathcal{R}_\varphi$ de l'application d'une expression XPath $\varphi(\bar{x})$ sur un document XML correspond à l'ensemble des tuples de liaison $\epsilon(\bar{x})$.

Exemple 17

Les tableaux $\mathcal{R}_{\varphi_i}$ contiennent tous les tuples de liaison générés pour les expressions XPath de l'exemple 15 appliquées au document WebLab 3.1.

$\mathcal{R}_{\varphi_1}$
$r
(2)

$\mathcal{R}_{\varphi_2}$	
$r	$x
(3)	*(4)*
(6)	*(7)*

$\mathcal{R}_{\varphi_3}$	
$r	$x
(3)	*(3)*

3.2.4 Évaluation des règles

L'application des règles de dépendance des données de type $\varphi_{S0}(\bar{x})[, \varphi_{Si}(\bar{x})]^* \rightarrow \varphi_{C0}(\bar{x})[, \varphi_{Ci}(\bar{x})]^*$ commence avec l'extraction des variables des expressions XPath d'entrée et de sortie. Une jointure est ensuite faite sur l'ensemble des variables explicites, les variables implicites étant renommées $\$in_i$ (respectivement $\$out_i$) pour l'expression à gauche (respectivement droite) de la flèche.

Définition 11 *Sémantique des règles*

Considérons la règle de dépendance $M = \varphi_{S_i}(\bar{x}) \rightarrow \varphi_{C_i}(\bar{x})$, son application sur le document XML d, notée $M(d)$, retourne un ensemble de liens de provenance sur les nœuds de d. $M(d)$ est définie par l'expression suivante appliquée à $\mathcal{R}_{\varphi_S}(d)$ et $\mathcal{R}_{\varphi_C}(d)$:

$$M(d) = \pi_{\$in_1,\$in_2\ldots,\$out_1,\$out_2,\ldots}(\mathcal{R}_{\varphi_{S_1}}(d) \bowtie \mathcal{R}_{\varphi_{S_2}}(d) \bowtie \ldots \bowtie \mathcal{R}_{\varphi_{C_1}}(d) \bowtie \mathcal{R}_{\varphi_{C_2}}(d) \bowtie \ldots)$$

Les deux opérations π et $\bowtie$ sont les opérations classiques de projection et de jointure d'algèbre relationnelle.

L'application de l'ensemble des règles associées à une exécution de service permet de générer le graphe de provenance que nous appelons $Graphe_{structurel}$:

Exemple 18

Les figures 3.5, 3.6, 3.7 et 3.8 montrent l'application de la règle de dépendance des données de l'extracteur de langue de l'exemple 16. Les tableaux de la figure 3.5 correspondent à l'application des fonctions des sous-expressions $\mathcal{R}_{\varphi_{(S|C)}}(d)$.

$\mathcal{R}_{\varphi_S}(d)$	
$\$in_1$	$\$x$
(4)	(3)
(7)	(6)

$\mathcal{R}_{\varphi_C}(d)$	
$\$out_1$	$\$x$
(5)	(3)
(8)	(6)

FIGURE 3.5 – Application des patrons XPath

Une jointure est ensuite faite entre les deux tableaux sur la variable $x, suivie d'une projection sur les variables $\$in_1$ *et* $\$out_1$ *pour ne plus voir que les liens de provenance dans le tableau 3.6.*

$\mathcal{R}_{\varphi_S}(d) \bowtie \mathcal{R}_{\varphi_T}(d)$		
$\$in_1$	$\$x$	$\$out_1$
(4)	(3)	(5)
(7)	(6)	(8)

$\pi_{\$in_1,\$out_1}(\mathcal{R}_{\varphi_S}(d) \bowtie \mathcal{R}_{\varphi_T}(d))$	
$\$in_1$	$\$out_1$
(4)	(5)
(7)	(8)

FIGURE 3.6 – Jointure des patrons XPath

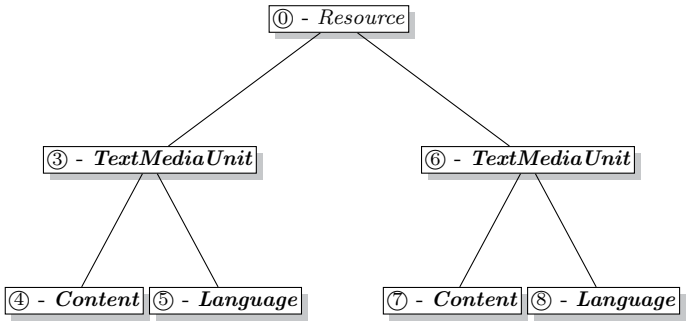


FIGURE 3.7 – Extrait du document

Le tableau montre deux liens de provenance entre les nœuds 4-5 et 7-8. On peut voir ici l'intérêt de l'utilisation des variables limitant les liens potentiels de provenance en évitant les combinaisons 4-8 et 7-5. Pour comparaison, le tableau de la figure 3.8 montre l'application de la même règle sans variable.

$//TextMediaUnit/Content \rightarrow //TextMediaUnit/Language$	
$\$in_1$	$\$out_1$
(4)	(5)
(4)	(8)
(7)	(5)
(7)	(8)

FIGURE 3.8 – Règle sans variable

3.2.5 Sémantique temporelle

Nous avons défini dans la section précédente 3.2.3 le graphe de provenance structurelle. Dans celui-ci, les traces d'exécution telles que l'ordre de création des informations n'étaient pas pris en compte. Dans cette section, nous allons définir ce qu'est un graphe temporel.

Exemple 19

Dans l'exemple précédent (exemple 18), deux liens de provenance sont générés alors que seul le lien entre les nœuds (4) et (5) est correct. Les nœuds (7) et (8) qui apparaissent dans le second lien ne sont pas présents dans le document après l'exécution de l'extracteur de langue, il n'y a donc aucune raison de créer ce lien. Nous pouvons voir sur la figure 3.2 que l'exécution de l'extracteur de langue a eu lieu à t3, créant l'unique nœud Language (5).

Pour éviter la création de liens incohérents, nous enregistrons différentes informations durant l'exécution de notre chaine de traitements media-mining. Ces informations incluent, pour chaque service exécuté, la date et heure de l'exécution, le service exécuté (et la règle de mapping associée), ainsi que les nouveaux nœuds générés. L'obtention de cette dernière information se fait par le moteur d'orchestration en comparant le document d'entrée et le document de sortie du service (fonction *diff*).

Connaitre les nœuds générés par une exécution de service permet d'affiner la granularité de la provenance en réduisant les liens de provenance superflus. Cela offre la possibilité d'appliquer chaque expression XPath sur l'état d'un document au moment de l'exécution d'un service.

Définition 12 *États de document*

Considérons la règle de dépendance $M = \varphi_S(\bar{x}) \rightarrow \varphi_T(\bar{x})$. Son application sur les états de documents d et d', notée $M(d, d')$, retourne un ensemble de liens de provenance entre les nœuds de d et les nœuds de d'. $M(d, d')$ peut être définie par la formule algébrique suivante appliquée aux tableaux des variables embarquées :

$$M(d, d') = \pi_{\$in_1,\ldots,\$out_1,\ldots}(\mathcal{R}_{\varphi_{S_1}}(d) \bowtie \ldots \bowtie \mathcal{R}_{\varphi_{C_1}}(d') \bowtie \ldots)$$

Il est à noter que l'état d'un document peut être reconstitué par une réécriture de la requête à l'aide d'annotations de date sur les nœuds. Il n'est donc pas nécessaire de reconstituer les documents en entrée et en sortie avant d'appliquer une règle de dépendance de service.

Exemple 20

Les tableaux et graphes suivants montrent l'application des règles de mapping de l'extracteur de langue et du traducteur sur le document WebLab. Les lignes filtrées par l'ajout des états de documents sont barrées ou en pointillées afin de voir les changements apportés par cette dernière fonctionnalité.

Extracteur de langue	
$in	$out
(4)	*(5)*
~~*(7)*~~	~~*(8)*~~

Traducteur	
$in	$out
~~*(3)*~~	~~*(3)*~~
(3)	*(6)*
~~*(6)*~~	~~*(3)*~~
~~*(6)*~~	~~*(6)*~~

3.3 Extensions

Bien que le modèle présenté possède l'ensemble des fonctionnalités requises pour notre problème, nous avons envisagé plusieurs améliorations à notre modèle afin d'offrir davantage de possibilités pour des requêtes plus précises.

Fonction de position Les expressions XPath permettent l'utilisation de la position d'un nœud par rapport à son parent. Cela peut être utilisé de deux manières. La première, par exemple, en reliant naïvement le $3^{ème}$ nœud au $4^{ème}$ de cette manière :

$$//A[3] \rightarrow //B[4]$$

La deuxième manière utilise la fonction *position()* de XPath et permet une utilisation de la position plus intéressante. Il devient ainsi possible de relier le $i^{ème}$ nœud A avec le $i^{ème} B$, pour l'ensemble des nœuds A et B :

$$//A[\$i = position()] \rightarrow //B[\$i = position()]$$

Fonction de Skolem Afin de définir des règles d'agrégations plus complexes au sein de notre système, il est envisageable d'utiliser des fonctions de Skolem. Ces fonctions sont utilisées dans une logique de prédicat afin de remplacer des variables quantifiées par des symboles de fonction.

Ces fonctions de Skolem pourraient être exploitées de plusieurs manières dans nos règles afin de définir différentes sortes d'agrégations, comme décrit dans [23]. Dans les exemples suivants, @a représente des attributs non identifiants et @id des attributs identifiants.

Un-vers-plusieurs :

$$//A[\$x = @uri] \rightarrow //B[f(\$x) = @a]$$

Dans cette règle, un unique nœud A est lié à plusieurs nœuds B possédant un attribut commun correspondant à la transformation de $x par la fonction f, comme montré sur la figure 3.9a.

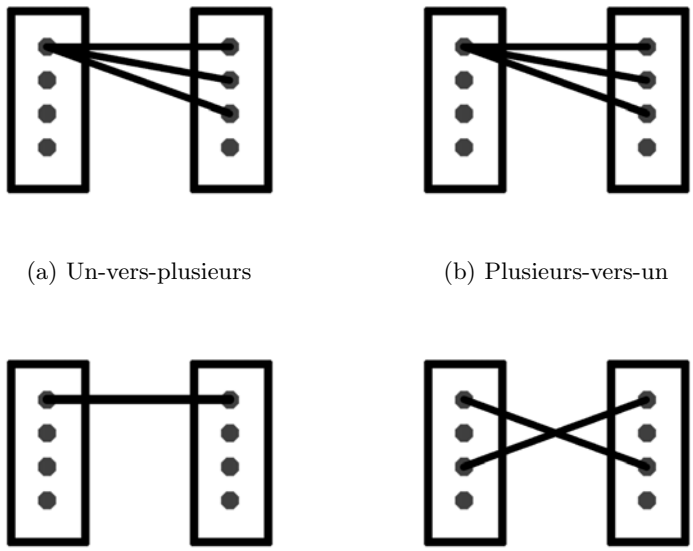
Un exemple simple, utilisant une fonction identité f($x) = $x, pourrait être cette règle :

$$//AudioMediaUnit[\$x = @uri] \rightarrow //TextMediaUnit[f(\$x) = @créé - par]$$

Ici, les unités de texte explicitent leurs liens de provenance, nous permettant d'utiliser l'information pour relier plusieurs unités à leur ancêtre commun.

Plusieurs-vers-un :

$$//AudioMediaUnit[\$x = @utilisé - par] \rightarrow //TextMediaUnit[f(\$x) = @uri]$$



(a) Un-vers-plusieurs (b) Plusieurs-vers-un

(c) Un-vers-un (d) Plusieurs-vers-plusieurs

FIGURE 3.9 – Fonctions de Skolem

Quand la règle *Un-vers-plusieurs* précédente reliait plusieurs TextMediaUnit vers un seul ancêtre, nous voyons ici l'inverse avec une unique TextMediaUnit reliée à plusieurs ancêtres 3.9b.

Un-vers-un :

$$//AudioMediaUnit[\$x = @uri] \rightarrow //TextMediaUnit[f(\$x) = @uri]$$

Ici chaque TextMediaUnit est reliée à une unique AudioMediaUnit 3.9c.

Plusieurs-vers-Plusieurs :

$$//AudioMediaUnit[\$x = @utilisé-par] \rightarrow //TextMediaUnit[f(\$x) = @créé-par]$$

Cette dernière règle permet de relier plusieurs TextMediaUnit à plusieurs AudioMediaUnit 3.9d.

3.4 Règles contextuelles

L'objectif premier de ce modèle est de proposer aux utilisateurs de chaines de traitement WebLab la possibilité de suivre la transformation des données au sein

d'une exécution. Une des contraintes du projet était que le modèle de provenance apporte le moins de modifications possible au système, particulièrement sur le temps d'exécution. Le modèle actuel enregistre quelques informations non-fonctionnelles pendant l'exécution de chaque service (date de l'exécution, services précédents et suivants, etc.), ainsi que les nouveaux nœuds créés au sein du document XML.

Le graphe de provenance est calculé a posteriori de l'exécution du workflow, appliquant les règles de dépendance de chaque service. La méthode naïve ici serait de recalculer à l'aide des méta-informations les états des documents en entrée et en sortie de l'exécution d'un service. La méthode serait facile à mettre en place mais très couteuse en temps.

Une autre méthode est l'ajout de contraintes au sein des règles, que nous appliquons ensuite sur un document final enrichi. Ce document est enrichi en associant les estampilles de temps de l'exécution d'un service à chaque nœud XML créé par ce même service, ainsi que le service exécuté.

Cette méthode demande d'ajouter quelques filtres aux expressions φ_S et φ_T, explicitement ou implicitement. Dans les prochaines expressions, l'attribut $@t$ correspond à la date de création du nœud, la valeur t est la date d'exécution du service, tandis que $@s$ et s sont respectivement à l'attribut donnant le service créateur du nœud, et la valeur du service exécuté pour cette règle.

Un filtre $[@t < t]$ est appliqué à chaque nœud final de φ_S, que ce soit le nœud final de l'expression, ou les nœuds finaux des filtres. Un autre filtre, $[@t = t \ and \ @s = s]$, est appliqué aux mêmes types de nœuds de φ_T. Le document est enrichi au préalable avec les attributs $@t$ et $@s$, obtenus lors de l'exécution des services.

Exemple 21

L'ajout de contraintes de temps et de services sur la règle de dépendance du traducteur transforme l'expression ainsi :

$$//TextMediaUnit[@t < t][//Language[@t < t]] \rightarrow //TextMediaUnit[@t = t \ and \ @s = s]$$

Une fois le graphe généré, il peut éventuellement être stocké afin d'éviter de le recalculer inutilement. Seuls les liens entre les nœuds (URI) ont besoin d'être enregistrés, permettant de limiter l'espace requis, les données étant déjà dans le document WebLab issu du traitement.

Une autre utilisation possible du modèle est la génération des liens de provenance au cours de l'exécution du workflow. L'idée ici est d'appliquer les règles de mapping

à la fin de l'exécution de chaque service, ayant connaissance à la fin du document d'entrée et du document de sortie.

Cette solution possède deux grands inconvénients : elle demande des ressources pendant l'exécution, augmentant donc le temps d'exécution de la chaine, et elle est très intrusive, demandant une modification conséquente du système d'orchestration.

Mais la génération des liens de provenance au cours de l'exécution ouvre de nouvelles possibilités, telles que celle d'avoir une chaine de traitements orientée sur la présence de données dans le document, ou la sélection d'un traducteur différent suivant la langue détectée pour un texte (sélection dynamique des services).

3.5 Conclusion

Dans ce chapitre, nous avons abordé et défini la notion de provenance. Nous avons présenté un modèle permettant la génération et le stockage de la provenance dans le cadre de la plateforme WebLab, ainsi que les moyens de l'améliorer.

Ce système est fondé sur un système de règles associées aux services WebLab permettant de générer les liens de dépendances entre les nœuds XML des documents WebLab. Les règles sont décrites sous un format inspiré de XPath et fonctionnent pour tout système fondé sur l'échange de documents XML. Le modèle collecte des métadonnées pendant l'exécution du workflow, puis génère un graphe de provenance a posteriori qui est enregistré dans un triplestore RDF au format standard W3C PROV.

Dans le chapitre suivant, nous abordons la gestion de la qualité au sein de la plateforme WebLab, en associant des règles de qualité à nos règles de provenance.

4 Gestion de la qualité

Nous proposons dans ce chapitre notre seconde contribution : un modèle de gestion de la qualité au sein de la plateforme WebLab. Ce modèle de propagation de la qualité dans un graphe est applicable sur n'importe quel graphe de provenance.

Dans la première section, nous présentons la manière dont nous stockons les informations de qualité au sein du modèle, ainsi que les différents domaines applicables aux dimensions de la qualité.

Dans la seconde section, nous présentons des règles, associées aux règles de provenance définies précédemment, permettant d'inférer des valeurs de qualité à des données à partir d'informations de qualité spécifiées par l'utilisateur.

4.1 Modèle d'annotation de qualité

Afin d'étendre notre modèle de provenance pour la gestion de la qualité des données, nous avons développé un modèle adapté pour l'annotation de qualité sur un nœud de données de type media-mining. Notre objectif ici n'est pas de définir des dimensions de qualité, mais de fournir un modèle suffisamment générique et ouvert pour être adapté à différents contextes.

Exemple 22

La chaine de traitements proposée en introduction (figure 1.1) génère et utilise différents types de données. Le tableau 4.1 montre un exemple de dimensions de qualité pour chaque type d'information du document WebLab (figure 1.2).

Nous trouvons dans ce tableau une colonne types de données, suivi d'une seconde colonne avec différentes dimensions de qualités applicables. Certaines dimensions sont communes à plusieurs types de données (exactitude, résolution, couleurs et taux). La dernière colonne affiche les valeurs de qualité possibles pour chaque dimension.

Certaines dimensions de qualité peuvent être mesurées directement (résolution d'une image, taux d'échantillonnage, etc.). La mesure d'autres dimensions telles que l'exactitude ou la complétude est plus complexe, et demande des informations supplémentaires. La complétude d'un ensemble d'entités peut, par exemple, être mesurée en comparant cet ensemble avec un ensemble idéal (connu ou estimé). Certaines mesures, comme la cohérence d'un texte, n'ont pas de définition formelle et peuvent nécessiter l'intervention humaine.

type de données	dimension de qualités	valeurs possibles
vidéo	résolution	[460x360, 640x480, 720x480, 800x600, 1024x768]
	couleurs majoritaires	rouge, vert, bleu, jaune, violet, orange, noir, blanc
	taux	98, 128, 256, ...
image	résolution	[460x360, 640x480, 720x480, 800x600, 1024x768]
	couleurs majoritaires	rouge, vert, bleu, jaune, violet, orange, noir, blanc
son	taux	98, 128, 256, ...
	bruit	[0,1]
texte	cohérence	forte, moyenne, faible
langage	exactitude	vrai, false
entité	exactitude	vrai, false
ensemble(entité)	complétude	[0,1]
	précision	[0,1]

FIGURE 4.1 – Exemple de dimensions de qualité

Exemple 23

Considérons à nouveau l'exemple du tableau 4.1. La couleur majoritaire d'une image ou d'une vidéo est une dimension dont le domaine de valeurs est l'ensemble $dom(couleur) = \{rouge, vert, bleu, jaune, violet, orange, noir, blanc\}$. *Cette dimension possède un domaine non-ordonné fini, il est possible ainsi de spécifier un sous-ensemble de valeurs de qualité* $\{rouge, orange, jaune\}$ *par trois annotations couleur(i) = rouge, couleur(i) = orange et couleur(i) = jaune.*

La résolution est définie sur un domaine ordonné fini non-continu, dont les valeurs possibles sont $\{460 \times 360, 640 \times 480, 720 \times 480, 800 \times 600, 1024 \times 768\}$ *(ordonnées sur le nombre total de pixels). Ce type de domaine permet à l'utilisateur de spécifier la qualité d'une donnée par comparaison avec une valeur. Par exemple, les annotations* $resolution(i) < 800 \times 600$ *et* $resolution(i) >= 640 \times 480$ *définissent comme intervalle de valeurs possibles* $\{640 \times 480, 720 \times 480\}$. *Il n'est cependant pas possible de spécifier* $resolution(i) <= 460 \times 360$ *et* $resolution(i) >= \{1024 \times 768\}$, *car ces annotations définissent deux intervalles disjoints.*

Le bruit d'un nœud de son est défini sur un domaine ordonné continu, dont les

valeurs sont dans l'intervalle de valeur réelle $[0, 1]$. *S'agissant d'un domaine ordonné, l'utilisateur peut estimer la qualité par comparaison avec une valeur. Il peut ainsi spécifier que le bruit d'une piste audio est entre* 0.3 *et* 0.5.

Définition 13 *Domaine de valeurs*

Soit Q *un ensemble de dimensions de qualité.*

Nous définissons pour chaque dimension de qualité $q \in Q$ *un ensemble de valeurs possibles que nous appelons domaine de* q, *ou* $dom(q)$. *Les domaines de qualité sont séparés en trois catégories d'ensemble : non-ordonné, ordonné non-continu, ordonné continu.*

- *Un domaine non-ordonné est un ensemble fini de valeurs. Ce type de domaine limite les comparaisons possibles entre domaine de qualité aux égalités* ($=$) *et inégalités* ($\neq$).
- *Un domaine ordonné non-continu est un ensemble fini de valeurs classées entre elles. Ce type de domaine permet les comparaisons de type égalité, d'inégalité, inférieur et supérieur* ($=,\neq,<,\leq,\geq,>$).
- *Un domaine ordonné continu est un ensemble infini de valeurs classées. Ce type de domaine permet les comparaisons* ($=,\neq,<,\leq,\geq,>$), *mais il n'est pas possible d'en énumérer les valeurs.*

Définition 14 *Qualité d'un noeud*

La qualité d'un nœud n *est définie sur une ou plusieurs dimensions. Chacune de ces dimensions appliquée à* n *est notée* $q(n)$, *et* $valeur(q, n) \subseteq dom(q)$ *est un ensemble de valeurs de la dimension de qualité* q *pour ce nœud. Nous considérons* $q(n)$ *comme inconnue si et seulement si* $value(q, n) = dom(q)$, *et incohérente si et seulement si* $value(q, n) = \emptyset$. $q(n)$ *est également considérée incohérente si le domaine de la qualité est ordonné et défini sur plusieurs intervalles disjoints.*

L'objectif de notre modèle est de pouvoir annoter des données par des estimations de leur qualité.

Définition 15 *Annotation de qualité*

Une annotation de qualité λ *pour une dimension* $\boldsymbol{q}$ *appliquée à un nœud* $\boldsymbol{n}$ *est une expression* $\lambda(q, n) = (q(n)\phi a)$ *où* ϕ *est un opérateur de comparaison de* $dom(q)$ *et* $a \in dom(q)$ *est une constante dans le domaine de* $\boldsymbol{q}$.

Toutes les annotations de qualité sont renseignées dans une table *A(nœud, prédicat)* contenant pour chaque nœud **n** les contraintes de qualité associées.

Définition 16 *Valeurs de qualité d'une annotation λ*

Soit Q un ensemble de dimensions et N un ensemble de nœuds, et $\lambda(q,n)$ l'annotation de la qualité d'un nœud $n \in N$ pour la dimension $q \in Q$.

Nous définissons $valeur(\lambda(q,n))$ le sous-ensemble de $dom(q)$ satisfaisant l'annotation $\lambda(q,n)$.

Définition 17 *Ensemble d'annotations A*

Soit A un ensemble d'annotations $\lambda_i(q,n)$ pour $q \in Q$ et $n \in N$

On note $A(q,n)$ le sous-ensemble des annotations $\lambda_i(q,n)$ dans A pour la dimension q et le nœud n.

Définition 18 *Valeurs de qualité en A*

*Nous définissons par $valeur(A(q,n)) = \cap_{\lambda \in A} valeur(\lambda(q,n))$ la valeur de la qualité d'un nœud **n** sur la dimension **q**, respectant les annotations $\lambda(q,n) \in A$.*

On peut montrer que $valeur(A(q,n))$ est le plus grand sous-ensemble de $dom(q)$ qui satisfait tous les $\lambda(q,n)$ dans $A(q,n)$.

Nous considérons **A** comme cohérent si et seulement si aucune valeur de qualité dans $valeur(A(q,n))$ n'est vide.

Exemple 24

En considérant toujours notre exemple issu de la figure 1.1, nous proposons le tableau d'annotations de qualité de la figure 4.2.

nœud	annotation
④	**cohérence** $< forte$
④	**cohérence** $> faible$
⑤	**exactitude** $= vrai$
⑦	$\emptyset$
⑨	**complétude** ≤ 0.8
⑨	**précision** < 0.9

FIGURE 4.2 – Annotations de qualité A

Cette table contient les annotations de qualité pour les nœuds ④, ⑤, ⑦ et ⑨ pour différentes dimensions de qualité. Pour chaque nœud, les valeurs de qualité satisfaisant les annotations de A sont montrées dans le tableau 4.3.

nœud	dimension	valeur
④	**cohérence**	$\{moyenne\}$
⑤	**exactitude**	$\{vrai\}$
⑦	**cohérence**	$\{faible, moyenne, forte\}$
⑨	**complétude**	$[0, 0.8]$
⑨	**précision**	$[0, 0.9[$

FIGURE 4.3 – Valeur(A)

Considérons par exemple la première ligne du tableau des valeurs de qualité ci-dessus affirmant que la cohérence du nœud ④ est moyenne, cette valeur est issue de la conjonction des deux premières lignes du tableau d'annotations de qualité : ***cohérence*** $< forte$ *et* ***cohérence*** $> faible$*. Or le domaine de cohérence ne répertoriant que les valeurs faible, moyenne et forte, l'unique valeur possible est moyenne.*

nœud	annotation
①	**cohérence** $>= forte$
①	**cohérence** $<= faible$

nœud	dimension	valeur
①	**cohérence**	$\emptyset$

FIGURE 4.4 – Exemple d'incohérence

Considérons maintenant les tableaux de la figure 4.4. Dans le tableau de gauche, les deux annotations proposées sont contradictoires, affirmant que la cohérence de la donnée ① doit être supérieure ou égale à forte, et inférieure ou égale à faible. Dans ce cas, les valeurs déduites pour le nœud forment un ensemble vide, visible dans le tableau de droite, et les annotations sont alors considérées comme incohérentes.

Proposition 1

Soit $A' \subseteq A$ deux ensembles d'annotations de qualité.

L'ensemble de valeurs $valeur(A)$ correspondant aux annotations de A est un sous-ensemble de $valeur(A')$.

Preuve : Si $A' \subseteq A$, alors l'intersection entre l'ensemble des valeurs des annotations $(A - A')$, notées $valeur(q, n, \lambda)$, et l'ensemble des valeurs des annotations de A' est nécessairement un sous-ensemble des valeurs de A. Ou, formellement : si $A' \subseteq A$, alors $valeur(q, n, A) = valeur(q, n, A - A') \cap valeur(q, n, A') \subseteq valeur(q, n, A)$.

4.2 Modèle d'inférence de qualité

Notre seconde contribution est un modèle permettant d'inférer de nouvelles valeurs de qualité aux ressources en utilisant les liens de provenance et le système d'annotations de la section précédente.

Afin de propager les valeurs de qualité dans le graphe de provenance, nous associons aux règles de provenance une ou plusieurs règles de qualité. Ces règles spécifient des contraintes de qualité entre les paramètres d'*entrée* et les paramètres de *sortie* de la règle de dépendance des données.

Définition 19 *Règles de qualité*

Soit M une règle de provenance entre les nœuds $\$IN = \{\$in1, \$in2, \ldots\}$ et les nœuds $\$OUT = \{\$out1, \$out2, \ldots\}$, respectivement les nœuds source et cible d'une règle de dépendance des données. Une règle de qualité pour le mapping M est définie avec une de ces deux formes :

1. *expression simple : $r = q(\$data)\phi a$ où $\$data \in \$IN \cup \$OUT$, $\phi \in \Phi(q)$ et $a \in dom(q)$*
2. *expression complexe : $r = q(\$in)\phi q(\$out)$ où $\$in \in \$IN$, $\$out \in \$OUT$ et $\phi \in \Phi(q)$*

Exemple 25

*Considérons un service de traduction possédant deux données d'entrée : le texte à traduire (ressource **Content**) et la langue du texte (**Language**), et une donnée de sortie : le texte traduit. La qualité des textes en entrée et en sortie est évaluée sur sa **cohérence**, tandis que la langue est évaluée sur son **exactitude**.*

Considérons la règle de dépendance de données M suivante :

$$\boldsymbol{R} :$$
$$//TextMediaUnit[\$x]/Content[\$in1], //TextMediaUnit[\$x]/Language[\$in2] \rightarrow //TextMediaUnit[\$out]$$

Cette règle de provenance liée au service de traduction crée des liens de provenance entre les nœuds sources $Content$ et $Language$ d'une unité de média, et un nœud cible $TextMediaUnit$. Les nœuds $Content$ et $Language$ sont respectivement appelés $\$in1$ et $\$in2$, représentant les entrées numéro 1 et 2. L'unique nœud cible $TextMediaUnit$ est appelé $\$out$.

Nous associons à la règle de dépendance de données $\boldsymbol{R}$ deux règles de qualité r_1 et r_2 :

TRADUCTEUR		
r_1	:	$\textbf{\textit{cohérence}}(\$in1) \geq \textbf{\textit{cohérence}}(\$out)$
r_2	:	$\textbf{\textit{exactitude}}(\$in2) = faux \rightarrow \textbf{\textit{cohérence}}(\$out) = faible$

La première règle, r_1, indique que la ***cohérence*** *du texte $out, issue de la traduction, possède une qualité toujours inférieure à celle du texte source.*

*La seconde règle, r_2, est une règle simple indiquant que si la langue identifiée est erronée (****exactitude*** *du nœud Language à faux), alors la* ***cohérence*** *du texte cible est toujours faible.*

Il faut maintenant instancier les règles avec les valeurs de qualité spécifiées par l'utilisateur ou inférées précédemment pour d'autres règles. Pour ce faire, les variables dans $\$IN$ et $\$OUT$ de chaque expression sont remplacées par les URIs de chaque liaison générée par les règles de provenance. Chaque règle ainsi instanciée crée de nouvelles contraintes de qualité dans le tableau des annotations de qualité, chaque nouvelle contrainte pouvant déclencher d'autres règles, jusqu'à la stabilisation du tableau (point fixe).

Définition 20 *Règle instanciée*

Soit M une règle de provenance et $liens(M, L) \subseteq L$ l'ensemble de liaisons entrées-sorties pour la règle de provenance M dans une table de lien L (cf. chapitre 3 page 35).

Soit règle $r \in règles(M)$ une règle de qualité associée au mapping M. Soit l une liaison dans $liens(M, L)$. La règle instanciée $instance(r, l)$ est obtenue par l'instanciation des paramètres $\$In$ et $\$Out$ du mapping M avec les valeurs de $liens(M, L)$.

L'ensemble des règles instanciées obtenues pour les règles $R = règles(M)$ et l'ensemble de liens L est noté $lien(R, L)$.

Exemple 26

On considère l'ensemble d'annotations de qualité A de l'exemple 24, la règle de qualité de l'exemple 25, et les liens L du tableau de provenance suivant, issus de la règle précédente appliquée sur les état d et d', respectivement les états avant et après l'exécution du service, du document de la figure 1.2.

L'instantiation des règles assigne la variable $\$in1$ au nœud ④, $\$in2$ à ⑤ et $\$out$ à ⑥. Le remplacement des variables dans les règles de qualités r_1 et r_2 donne deux nouvelles règles instanciées $instance(r_1, l)$ et $instance(r_2, l)$.

l=

$M = //TextMediaUnit/Content, //TextMediaUnit/Language \rightarrow //TextMediaUnit$		
$\$in1$	$\$in2$	$\$out$
④	⑤	⑥

FIGURE 4.5 – liens(M,L)

TRADUCTEUR		
$instance(r_1, l)$	:	$\textbf{\textit{cohérence}}(④) \geq \textbf{\textit{cohérence}}(⑥)$
$instance(r_2, l)$	:	$\textbf{\textit{exactitude}}(⑤) = faux \rightarrow \textbf{\textit{cohérence}}(⑥) = faible$

Le résultat de l'application des règles instanciées sur les valeurs de qualités initiales est défini comme ceci :

Définition 21 *Résultat*

Le résultat d'un ensemble de règles de qualité R, respectant un ensemble d'annotations A et des liens de provenance L, est l'ensemble de valeurs de qualité satisfaisant les annotations de A et les règles instanciées $Instances(R, L)$.

Exemple 27

Considérons $R = \{r_1, r_2\}$, A et L des exemples précédents. La valeur de qualité ***cohérence***$(④) = \{moyenne\}$ *et la règle $lien(r_1, M)$ apporte une nouvelle contrainte à la table d'annotation A :* ***cohérence***$(⑥) = \{faible, moyenne\}$*. La valeur haute n'est plus considérée possible pour le nœud ⑥ car la* ***cohérence*** *de ce dernier doit être inférieure ou égale à la* ***cohérence*** *du nœud ④, fixée à moyenne. La règle r_2 n'a aucun effet ici étant donné que l'****exactitude*** *du nœud ⑤ est égale à vrai.*

Algorithme d'évaluation de règle de qualité

L'algorithme 1 compile un ensemble d'annotations de qualité V à partir de valeurs de qualité initiales $valeur(A)$ en appliquant de manière itérative un ensemble de règles instanciées de qualité avec les liens de L.

Algorithm 1: Évaluation de règles

entrée: règles de qualité R, table d'annotation de qualité A, liens de provenance L

sortie : table de valeurs de qualité V

1 $V = valeur(A)$;
2 **répéter**
3 **pour** chaque lien de provenance $l \in L$ **faire**
4 **pour** chaque règle $P \rightarrow Q \in règles(L)$ **faire**
5 $PV := instance(P, L)$;
6 **si** $cohérent(PV, V)$ **alors**
7 $QV := instance(Q, L)$;
8 $intersection(V, QV)$;
9 **pour** $\mathbf{q}(\$in) \phi\, \mathbf{q}(\$out) \in règles(s)$ **faire**
10 $X := instance(\mathbf{q}(\$in), l)$;
11 $Y := instance(\mathbf{q}(\$out), l)$;
12 $UV := valeur(X\, \phi\, valeur(Y, V))$;
13 $intersection(V, UV)$;
14 $UV := valeur(valeur(X, V)\, \phi\, Y)$;
15 $intersection(V, UV)$;
16 **jusqu'à** V ne change plus;

La ligne 1 initialise toutes les valeurs de qualité V avec les valeurs $valeur(A)$ correspondant aux annotations de qualité de A. Nous supposons que A contient pour tous les nœuds et toutes les dimensions de qualité correspondantes $\mathbf{q}$ une annotation par défaut $\mathbf{q}(\text{ⓝ}) = dom(\mathbf{q})$.

La boucle suivante (lignes 2-16) s'arrête quand toutes les règles de qualité de chaque lien de provenance L n'ont plus aucun effet sur les valeurs de qualité V (valeurs arrivées à un point fixe).

Ensuite, les deux types de règles de qualité sont évalués séparément. La première boucle (lignes 4-8) ajoute les valeurs générées par l'ensemble des règles instanciées $lien(Q, l)$ à l'ensemble V si les valeurs générées par $instance(P, l)$ sont cohérentes avec les valeurs existantes de V. L'opération $intersection(V, QV)$ remplace chaque valeur de qualité de V par l'intersection avec les valeurs correspondantes QV (pour le même nœud et la même dimension).

La seconde boucle (lignes 9-15) remplace les expressions $\mathbf{q}(\$in)$ et $\mathbf{q}(\$out)$ par les nœuds correspondant de L. Les expressions $valeur(X, V)$ et $valeur(Y, V)$ extraient les valeurs de qualité de V pour les nœuds précédents. Ces valeurs sont ensuite utilisées pour construire les nouvelles contraintes qui remplaceront les valeurs de qualité de V correspondantes par l'utilisation de la fonction d'intersection.

Exemple 28

L'application de la première boucle (lignes 2-16) à notre traducteur ajoute les règles ***cohérence****($in1) ≥* ***cohérence****($out) et* ***exactitude****($in2)=faux →* ***cohérence****($out) = faible aux valeurs de qualités V.*

La seconde boucle (lignes 9-15) remplacera les entrées $\$in1, \$in2, \$out$ par les nœuds correspondants {④, ⑤, ⑥}, obtenant ainsi les règles ***cohérence****(④) ≥* ***cohérence****(⑥) et* ***exactitude****(⑤)=faux →* ***cohérence****(⑥) = faible.*

Cet algorithme modifie l'ensemble des valeurs de qualité V simplement en remplaçant les valeurs existantes par l'intersection avec les nouvelles valeurs (lignes 8,13 et 15). On voit que l'algorithme est monotone et converge vers un ensemble final de valeurs de qualité (point fixe) après un nombre fini d'étapes.

Par analogie avec d'autres langages déductifs sans négation, ce point fixe est unique, et indépendant de l'ordre d'exécution des règles.

Perspective

Définition 22 *Chemin relatif*

Soit M une règle de provenance entre les nœuds $\$IN = \{\$in1, \$in2, \ldots\}$ et les nœuds $\$IN = \{\$in1, \$in2, \ldots\}$, et R un ensemble de règles de qualité associé. Chaque variable $\$in_i$ et $\$out_i$, définissant un nœud dans la règle de provenance, peut préciser un sous-nœud sur lequel s'applique la règle de qualité en spécifiant un chemin relatif XPath.

Exemple 29

Nous avons considéré précédemment le couple de règles suivant par l'exécution d'un traducteur.

TRADUCTEUR		
M	:	$//TextMediaUnit[\$x]/Content, //TextMediaUnit[\$x]/Language \rightarrow //TextMediaUnit$
r_1	:	$\mathbf{cohérence}(\$in1) \geq \mathbf{cohérence}(\$out)$
r_2	:	$\mathbf{exactitude}(\$in2) = faux \rightarrow \mathbf{cohérence}(\$out) = faible$

Il est possible de simplifier la règle de provenance en ne spécifiant qu'un lien entre les deux TextMediaUnit, et en précisant les sous-noeuds dans les règles de qualité :

TRADUCTEUR		
M	:	$//TextMediaUnit \rightarrow //TextMediaUnit$
r_1	:	$\textbf{\textit{cohérence}}(\$in/Content) \geq \textbf{\textit{cohérence}}(\$out)$
r_2	:	$\textbf{\textit{exactitude}}(\$in/Language) = faux \rightarrow \textbf{\textit{cohérence}}(\$out) = faible$

Cela permet également de spécifier la qualité plus précisément en identifiant un sous-nœud de celui proposé par la règle de provenance.

TRADUCTEUR		
M	:	$//TextMediaUnit \rightarrow //TextMediaUnit$
r_1	:	$\textbf{\textit{cohérence}}(\$in/Content) \geq \textbf{\textit{cohérence}}(\$out/Content)$
r_2	:	$\textbf{\textit{exactitude}}(\$in/Language) = faux$ $\rightarrow \textbf{\textit{cohérence}}(\$out/Content) = faible$

Le tableau ci-dessus précise que la ***cohérence*** *de la sortie du service est liée au sous-nœud Content, à la place du nœud TextMediaUnit.*

4.3 Conclusion

Dans ce chapitre, nous avons abordé et défini la notion de qualité associée à un graphe de provenance. Nous avons également présenté un modèle permettant la propagation de la qualité au sein de ce type de graphe, ainsi qu'une méthode de stockage compatible avec le modèle de provenance et adaptée à la plateforme WebLab.

Ce système est fondé sur un système de règles de qualité associées aux règles de provenance. Les règles sont décrites sous forme de comparaison de dimensions de qualité entre une ou plusieurs données d'entrée et une ou plusieurs données de sortie.

Dans le chapitre suivant, nous montrons la réalisation d'un prototype implémentant nos modèles de provenance et de qualité, sous la forme d'une IHM permettant aux utilisateurs de définir leurs propres règles.

5

Réalisation

Nous avons développé un prototype implémentant nos solutions présentées dans les chapitres 3 et 4. L'architecture globale de ce prototype est visible sur la figure 5.1.

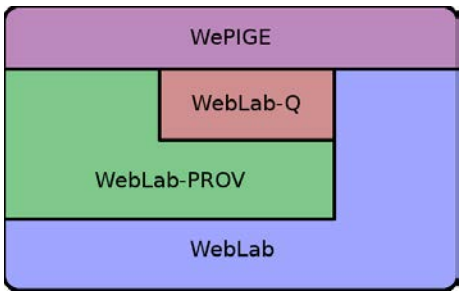


FIGURE 5.1 – Architecture globale

Cette architecture est composée de 4 couches : la plateforme **WebLab** , le moteur de provenance **WebLab-PROV**, le module de qualité **WebLab-Q**, et enfin l'interface homme-machine **WePIGE**. Une architecture plus complète est disponible plus tard (figure 5.15 page 81).

La plateforme WebLab exécute des services media-mining qui échangent des documents XML entre eux. **WebLab-PROV** est l'implémentation du modèle de provenance présenté dans le chapitre 3, et **WebLab-Q** celle de notre modèle de qualité présenté dans le chapitre 4. Enfin, **WePIGE** est l'interface utilisateur permettant d'accéder aux, et interagir avec, les informations de provenance et de qualité.

Nous présentons plus en détails ces trois modules dans les sections suivantes.

5.1 WebLab-PROV

La figure 5.2 montre l'architecture du module de provenance WebLab-PROV. Le composant récupère les traces d'exécutions générées par l'orchestrateur de services, le document XML final de l'exécution du workflow, ainsi que les mappings de service stockés dans un catalogue de services. Les règles de dépendances sont ensuite transformées, puis combinées avec les informations d'exécutions afin d'être exécutées sur le document XML pour générer un graphe de provenance qui sera stocké dans un triplestore RDF. Nous allons détailler chaque étape par la suite.

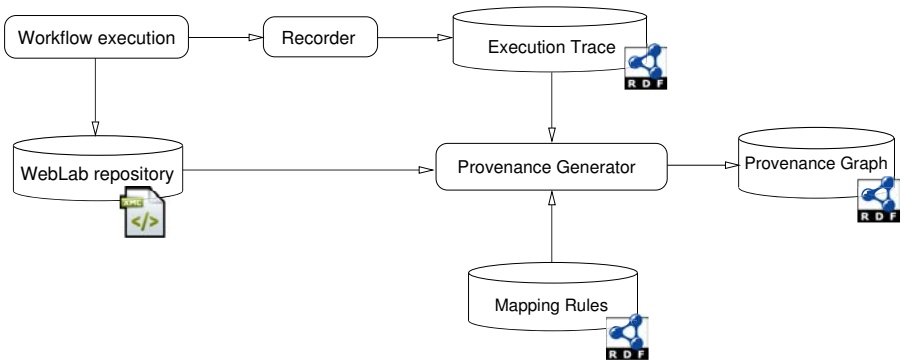


FIGURE 5.2 – WebLab-PROV : architecture fonctionnelle

5.1.1 Modèle d'enregistrement

Les informations de traces sont enregistrées dans une base de connaissances RDF en accord avec les *Starting Point Terms* de l'ontologie PROV [1]. Chaque exécution de service crée une instance de prov:Activity, et deux instances de prov:Entity comme le montre la figure 5.3.

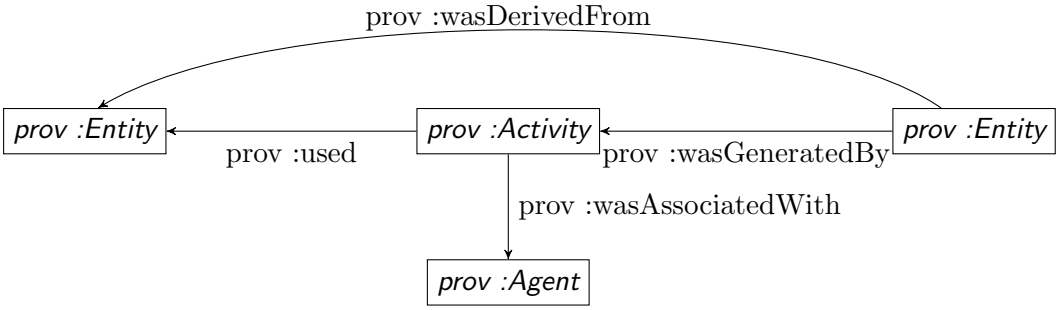


FIGURE 5.3 – Modèle RDF

Chaque triplet dont le prédicat est préfixé par **prov** fait partie du modèle *W3C Prov*. Nous avons étendu ce modèle avec des nouveaux prédicats avec le préfixe *wlprov* pour WebLab Prov, afin d'ajouter des liens vers les services précédents et suivants, ainsi que vers l'identifiant du workflow exécuté.

Exemple 30

Un exemple d'enregistrement au format RDF pour l'exécution d'un service est visible sur la figure 5.4. Cet enregistrement contient les informations de l'exécution d'un traducteur de 00 :00 à 00 :02 transformant un texte en texte-traduit. Ce service

1. http ://www.w3.org/TR/prov-o/

```
Traces d'exécution :

  'execution' prov:startedAtTime '00:00'
  'execution' prov:endedAtTime '00:02'
  'execution' prov:wasAssociatedWith 'traducteur'
  'execution' prov:used 'texte'
  'texte-traduit' prov:wasGeneratedBy 'execution'
  'execution' wlprov:previous 'execution-extracteur-de-langue'
  'execution' wlprov:next 'execution-extracteur-d-entités'
  'execution' wlprov:id 'mon-workflow'
```

FIGURE 5.4 – Exemple d'enregistrement RDF : Activity

```
  'traducteur' owls:serviceName 'traducteur anglais-français'
  'traducteur' owls:contactInformation 'http://traducteur?wsdl'
  'traducteur' wlprov:hasMapping '//TextMediaUnit[$x]/Content,
                          //TextMediaUnit[$x]/Language -> //TextMediaUnit'
```

FIGURE 5.5 – Exemple d'enregistrement RDF : Agent

est exécuté au sein d'un workflow identifié mon-workflow, entre un extracteur de langue et un extracteur d'entités nommées.

Catalogue de services

L'objet prov:Agent correspond à une instance d'un service (composant), contenant toutes les informations nécessaires à son identification et à son appel, ainsi qu'à l'identification des données d'entrée et de sortie.

L'ensemble des agents renseignés constitue ce que nous appelons un catalogue de services. Les informations de chaque service sont renseignées par les fournisseurs de services au moment de l'intégration du service dans la plateforme.

Exemple 31

Un exemple d'enregistrement au format RDF pour un service est visible sur la figure 5.5. Cet enregistrement contient les informations d'un traducteur, avec l'adresse pour contacter le service (http://traducteur?wsdl), ainsi que la règle de dépendance des données liée au traducteur.

Entrées / Sorties

L'objet prov:Entity contient les informations sur les entrées et sorties de l'exécution d'un service. Les données de sortie sont renseignées après l'invocation du service, les données d'entrée seront renseignées lors de l'application des règles de mapping.

Exemple 32

Un exemple d'enregistrement au format RDF pour des données est visible sur la figure 5.6. Cet enregistrement contient les informations de l'exécution d'un traducteur.

```
'traducteur' prov:used 'texte'
'traducteur' prov:used 'texte2'
'texte-traduit' prov:wasGeneratedBy 'traducteur'
'texte-traduit2' prov:wasGeneratedBy 'traducteur'

'texte' wlprov:correspondTo 'weblab://texte'
'texte' wlprov:correspondTo 'weblab://texte2'
'texte-traduit' wlprov:correspondTo 'weblab://texte-traduit'
'texte-traduit2' wlprov:correspondTo 'weblab://texte-traduit2'

'texte-traduit' prov:wasDerivedFrom 'texte'
'texte-traduit2' prov:wasDerivedFrom 'texte2'
```

FIGURE 5.6 – Exemple d'enregistrement RDF : Entity

Cet enregistrement contient deux ressources en sortie (texte-traduit et texte-traduit2), et deux ressources en entrée (texte et texte2), spécifiées par les quatre premières lignes de la figure 5.6. Les quatre lignes suivantes spécifient les ressources XML auxquelles sont liées les entités, et les deux dernières affinent les dépendances entre les données en spécifiant que l'entité texte-traduit est issue d'une transformation de l'entité texte, et que l'entité texte-traduit2 est issue de l'entité texte2.

Le prédicat *wlprov:correspondTo* donne l'URI de la ressource XML correspondante dans le document WebLab. Le prédicat *prov:wasDerivedFrom* donne l'information de l'objet source de la ressource courante. Il permet de définir des informations de provenance plus fines que l'utilisation de *prov:used* et *prov:wasGeneratedBy* sur l'entité *prov:Activity*. En effet, *prov:wasDerivedFrom* permet de lier un sous-ensemble de nœuds dérivés à un sous-ensemble de nœuds sources, tandis que *prov:used* et *prov:wasGeneratedBy* ne permettent de lier que l'ensemble des nœuds dérivés à l'ensemble des nœuds sources.

À partir de ces informations liées à l'exécution d'un service, nous allons maintenant voir comment utiliser les règles de mapping que nous avons définies précédemment pour générer des graphes de provenance.

5.1.2 Enregistrement des traces d'exécution

La première étape de la génération du graphe de provenance consiste en l'enregistrement des informations générées au cours de l'exécution d'un workflow. Nous avons modifié le système d'orchestration des services afin d'extraire les informations suivantes pour chaque appel de service : date d'exécution, durée d'exécution, service exécuté, données créées, services précédents, services suivants.

L'ensemble des données est enregistré par l'orchestrateur pendant l'exécution. Les données *date* et *durée* sont calculées pendant l'invocation d'un service, tandis que les données *service exécuté*, *services précédents* et *services suivants* sont renseignées à l'aide du control flow géré par l'orchestrateur.

La génération de l'information concernant les données créées pendant un appel de service demande un traitement plus complexe. En effet, le fonctionnement des services étant inconnu (services dits *boites noires*), nous n'avons aucune information concernant les données créées par un service. En exploitant la contrainte WebLab imposant qu'un service puisse uniquement ajouter des connaissances, il est possible de déduire les données créées en comparant le document XML en entrée avec le document XML en sortie de l'appel.

FIGURE 5.7 – Documents d'entrée et de sortie

La figure 5.7 montre les documents d'entrée et de sortie d'une exécution de service. L'objectif étant d'identifier les nœuds créés par le service, l'orchestrateur compare les versions du document WebLab avant et après l'exécution d'un service, puis identifie les ressources générées par le service. Cette opération est correcte car seules les insertions sont autorisées dans le modèle WebLab. La fonction de différence appliquée à notre

exemple permet de recueillir l'URI du nœud Entities ③. Ce traitement peut être couteux et dépend de la taille du document XML. Néanmoins, nos expériences ont montré que le temps de traitement demandé par cette opération est négligeable dans le contexte d'invocation de services de traitement de medias.

5.1.3 Génération du graphe de provenance

La génération du graphe de provenance se fait par l'application de l'algorithme 2 (page 70) aux règles de provenance à chaque exécution de service. Chaque application génère un tableau de liens entre les nœuds sources et dérivés.

Algorithm 2: Génération de la provenance

entrée: données d'exécution de workflow w, document final d

sortie : liens de provenance L

```
1  pour chaque service s de w faire
2      t := temps(s);
3      R := sortie(s);
4      annoter − ressources − document(d, t, R);
5  pour chaque service s de w faire
6      t := temps(s);
7      pour chaque mapping m de s faire
8          x := transformation(m);
9          L := application − regle(d, x, t);
10         enregistrer(L);
```

Cet algorithme fait deux passes pour chaque service exécuté dans le workflow. Dans la première boucle (ligne 1), l'algorithme recense l'ensemble des ressources créées par le service, ainsi que l'estampille de temps liée à cette exécution, afin d'annoter les nouvelles ressources du document avec la date de leurs créations (ligne 4).

La seconde boucle (ligne 5) transforme chaque règle liée à un service dans un langage de requête XML commun (XQuery), puis applique cette requête au document XML annoté. Le résultat est un ensemble de liens de provenance entre les ressources enregistrées dans la base de connaissance RDF.

Exemple 33

Considérons le document WebLab de la figure 5.8. La première boucle de l'algorithme 2 interroge les traces de l'exécution afin de répertorier les temps de création

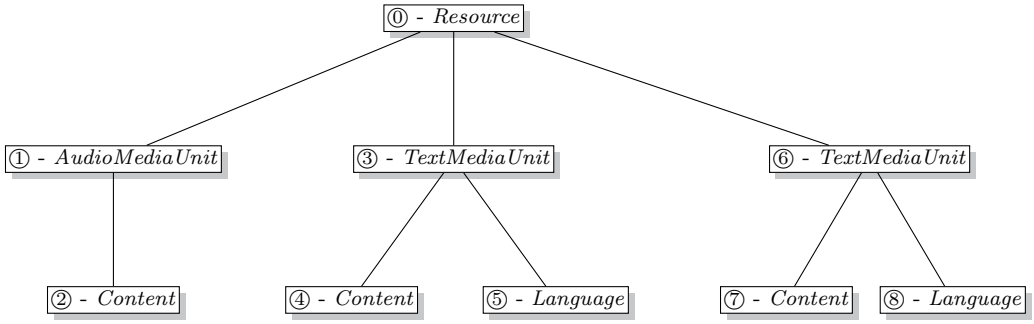


FIGURE 5.8 – Document final

de chaque ressource, puis ajoute ces informations au document XML. Dans notre exemple, les noeuds ③, ⑤ et ⑥ créés respectivement à t1 par le transcripteur, t2 par l'extracteur de langue et t3 par le traducteur (figure 5.9).

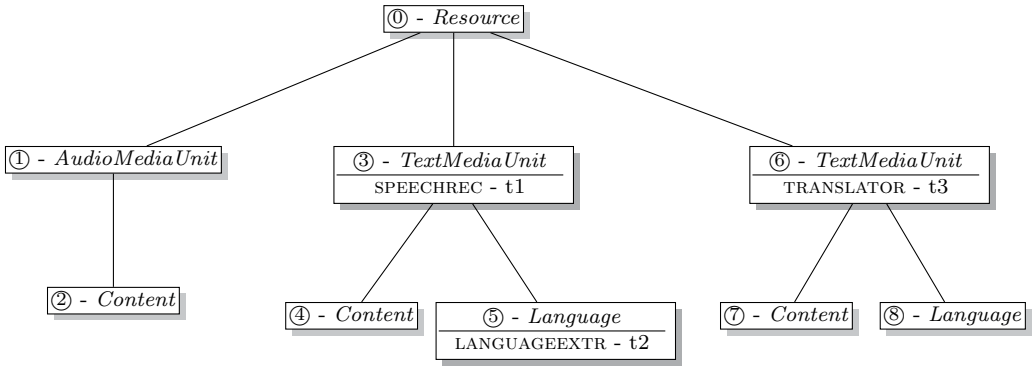


FIGURE 5.9 – Document annoté

La seconde boucle applique les règles de dépendance des données des services :
Transcripteur : $/resource/AudioMediaUnit/Content \rightarrow //TextMediaUnit$
Extracteur de langue : $//TextMediaUnit[\$x]/Content \rightarrow //TextMediaUnit[\$x]/Language$
Traducteur : $//TextMediaUnit[\$x]/Content, //TextMediaUnit[\$x]/Language \rightarrow //TextMediaUnit$

Chaque règle est transformée en requête XQuery, et filtrée avec la variable de temps correspondant à l'exécution du service. Ainsi, la transformation de la première règle par le compilateur donne l'expression XQuery suivante :

```
for $source in /resource/AudioMediaUnit/Content,
    $derive in //TextMediaUnit
where $source/@timestamp < t1
and $derive/@timestamp = t1
```

```
return <link>$derive prov:wasDerivedFrom $source</link>
```

La transformation des requêtes suit le principe suivant : chaque chemin XPath crée une liste de noeud XML stockée dans une variable ($source et $derive). Cette liste est ensuite filtrée par rapport au timestamp enregistré durant l'exécution : seuls les nœuds $source antérieurs à l'exécution, et seuls les nœuds $derive créés lors de cette exécution, sont gardés. Un lien de provenance est finalement créé entre chaque noeud $source et $derive restant. Cette règle convient donc pour représenter un traducteur, sachant que le fonctionnement standard d'un tel composant au sein de la plateforme WebLab est de traduire toutes les MediaUnits présentes.

La requête appliquée au document XML de la figure 5.8 retourne comme résultat une ligne XML <link>, donc la représentation sous forme de tableau de liens est la suivante :

source	*dérivé*
②	③

La même méthode est utilisée pour les deux autres exécutions de service, retournant ainsi les requêtes XQuery et tableaux de liens suivants :

```
for $source in //TextMediaUnit, $derive in //TextMediaUnit
let $x :=$source/@uri
where $source/Content and $derive/Language
and $derive/@uri = $x and $source/Content/@timestamp < t2
and $derive/Language/@timestamp = t2
return
    <link>$derive/Language prov:wasDerivedFrom $source/Content</link>
```

source	*dérivé*
④	⑤

```
for $source1 in //TextMediaUnit, $source2 in //TextMediaUnit,
    $derive in //TextMediaUnit
let $x :=$source1/@uri
where $source2/@uri = $x and $source1/Content
and $source2/Language and $source2/@timestamp < t3
and $source1/@timestamp < t3 and $derive/@timestamp = t3
return <link>$derive prov:wasDerivedFrom $source1</link>
<link>$derive prov:wasDerivedFrom $source2</link>
```

source	*dérivé*
④	⑥
⑤	⑥

Le graphe de provenance obtenu correspond au résultat illustré sur la figure 3.3 page 38. Ce graphe est enregistré dans une base de connaissance RDF et sert de base à notre système de gestion de la qualité.

5.1.4 Traducteur de règles

Nous avons développé un outil de transformation de nos règles de dépendance des données dans le langage XQuery.

XQuery [12] est un langage de requête fonctionnel (Turing-complet) pour extraire des informations d'un document ou d'une collection de documents XML, effectuer des calculs complexes à partir des informations extraites, et reconstruire de nouveaux documents XML.

L'outil de transformation utilise JavaCC [47] (Java Compiler Compiler, Compilateur de compilateur) pour analyser les règles de provenance et en déduire un parseur pour transformer les règles en expressions XQuery qui génèrent les liens de dépendances définis par les règles.

Un extrait de la grammaire du parseur est visible sur la figure 5.10. La première ligne déclare deux chaines de caractères *clés*, la virgule délimitant les patrons XPath entre eux, et la flèche délimitant les éléments sources et dérivés.

Après les déclarations et l'initialisation des variables, le premier élément *xpathPattern = createXpathPattern()* est traité (ligne 9). Cet élément est un patron XPath source traité dans une fonction séparée, dont la présence est obligatoire. Vient ensuite l'ensemble *(<COMA>[...])** traitant les patrons XPath sources optionnels séparés par une virgule. Le caractère $*$ présent après les parenthèses définit une occurrence de cette expression entre 0 et n, avec $n > 0$.

Dans la suite de l'expression, la flèche "->" sépare la partie gauche de la partie droite de la règle. Un premier patron XPath dérivé obligatoire s'ensuit, puis les patrons optionnels séparés par une virgule, de la même manière que précédemment. Si la règle ne correspond pas à la grammaire (deux virgules à la suite par exemple), alors elle est considérée erronée et rejetée.

Cette grammaire permet de créer l'arbre syntaxique présenté sur la figure 5.11. Chaque branche de l'arbre correspond à un élément obligatoire, à l'exception de

```
< COMA : "," > | < TO : "->" >

Element createRule() :
{
  MappingElement result = new MappingElement();
  XpathElement xpathPattern;
}
{
  xpathPattern = createXpathPattern()
  {
    result.addFrom(xpathPattern);
  }
  (
    < COMA >
    xpathPattern = createXpathPattern()
    {
      result.addFrom(xpathPattern);
    }
  )*
  < TO >
  xpathPattern = createXpathPattern()
  {
    result.addTo(xpathPattern);
  }
  (
    < COMA >
    xpathPattern = createXpathPattern()
    {
      result.addTo(xpathPattern);
    }
  )*
  {
    return result;
  }
}
```

FIGURE 5.10 – Extrait de grammaire JavaCC

celles annotées par une $*$, rendant le sous-arbre optionnel et pouvant être présent plusieurs fois. Cet arbre est ensuite analysé de manière à créer une expression XQuery à appliquer au document XML WebLab. Durant cette étape, le compilateur ajoute des contraintes temporelles et sémantiques (un nœud ne peut dépendre que d'un nœud créé antérieurement, les nœuds dérivés doivent avoir été créés durant l'exécution du service auquel la règle est appliquée, etc.).

Exemple 34

Considérons l'appel d'un service d'extraction de langue à un instant t, et la règle de provenance :

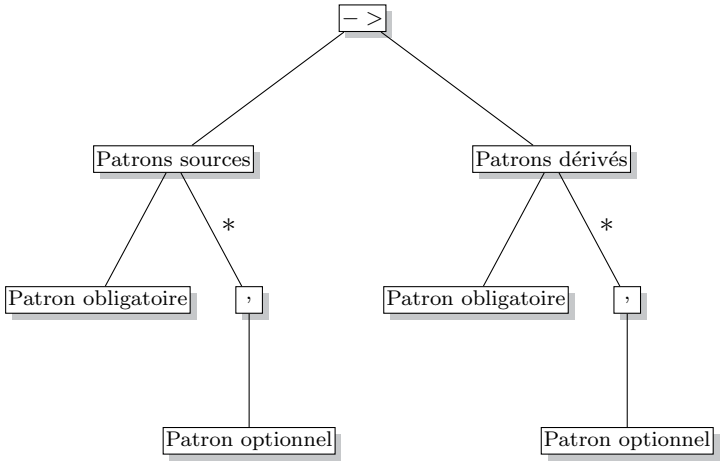


FIGURE 5.11 – Grammaire WebLab-PROV

$$//TextMediaUnit[\$x]/Content- > //TextMediaUnit[\$x]/Language$$

L'application de notre compilateur crée la requête XQuery suivante, qui peut être directement appliquée au document XML pour générer une séquence de liens de provenance sous la forme d'éléments <link> :

```
for $source in //TextMediaUnit, $derive in //TextMediaUnit
let $x :=$source/@uri
where $source/Content
and $derive/Language and $derive/@uri = $x
and $source/Content/@timestamp < t
and $derive/Language/@timestamp = t
return
    <link>$derive/Language prov:wasDerivedFrom $source/Content</link>
```

Nous pouvons voir dans la requête la présence de conditions qui comparent l'attribut timestamp à un paramètre t. Le timestamp correspond à l'annotation de temps ajoutée dans la première boucle de l'algorithme 2. Cette annotation est comparée à l'estampille de temps t correspondant à l'exécution du service liée à ce mapping.

5.2 WebLab-Quality

Dans cette section, nous allons voir comment les règles de qualité définies conjointement avec les règles de provenance (chapitre 4 page 53) sont appliquées au modèle WebLab. De la même manière que les règles de provenance, elles sont transformées

dans un langage permettant leur application. Nous avons choisi ici d'utiliser des règles d'inférence RDF.

Les règles de qualité sont transformées et appliquées aux liens identifiés par le modèle de provenance précédent. Les règles sont traduites dans un langage d'inférence traité par le triplestore dans lequel le graphe de provenance est enregistré, afin de gérer un système de contraintes sur les valeurs de qualité de chaque ressource WebLab enregistrée.

Nous avons utilisé Apache Jena comme base de connaissance RDF de manière à profiter du moteur d'inférence Jena. L'avantage de l'utilisation d'un moteur d'inférence RDF est l'application automatique des règles dès l'ajout de nouvelles informations. De plus, le modèle Jena fournit diverses fonctions d'agrégation (min, max, etc.) et de comparaison (greaterThan, lowerThan), qui permettent de créer nos règles facilement.

Il est à noter que dans cette section, le prédicat *:hasInput* est identique à *prov:used*, et le prédicat *:hasOutput* est l'inverse de *prov:used*.

Exemple 35

Une règle d'inférence Jena ressemble à ceci :

```
[nom-regle:
(ex:A ex:predicate ex:B), (ex:B ex:predicate ex:C)
                        -> (ex:A ex:predicate ex:C)
]
```

Cette règle est automatiquement exécutée lorsque les triplets "ex:A ex:predicate ex:B" et "ex:B ex:predicate ex:C" sont présents dans la base de connaissances. Elle crée ainsi un triplet "ex:A ex:predicate ex:C" à partir des deux triplets précédents.

Nous avons à nouveau utilisé JavaCC pour la transformation des règles de qualité au modèle Jena. Le modèle de qualité présenté dans le chapitre 4 considère des domaines ordonnés et non-ordonnés. Nous construisons nos règles Jena en considérant, pour chaque dimension de qualité, un intervalle [min:max] dans le domaine ordonné correspondant au type de la dimension de qualité (entier, flottant, etc.), ou un ensemble de valeurs pour les domaines non-ordonnés.

Exemple 36

Nous pouvons voir sur la figure 5.12 la règle de provenance $R1$ et les deux règles de qualité $Q1$ et $Q2$ du traducteur. La règle $Q1$ est traduite dans le langage Jena de la manière suivante :

Règle de provenance (R1)
$//TextMediaUnit[\$x]/Content, //TextMediaUnit[\$x]/Language \rightarrow //TextMediaUnit$

Règles de qualité
(Q1) $Consistency(\$in1) \geq Consistency(\$out)$
(Q2) $Correct(\$in2) = false \rightarrow Consistency(\$out) \leq basse$

FIGURE 5.12 – Récapitulatif des règles du traducteur

```
[translator-consistency:
(:translator :hasInput ?i), (:translator :hasOutput ?o),
(?i :consistency ?cons1), (?o :consistency ?cons2),
(?i :createdByRule '1-1'), (?o :createdByRule '1-3'),
(?cons1 :max ?value) -> (?cons2 :max ?value)
]
[translator-consistency-inverse:
(:translator :hasInput ?i), (:translator :hasOutput ?o),
(?i :createdByRule '1-1'), (?o :createdByRule '1-3'),
(?i :consistency ?cons1), (?o :consistency ?cons2),
(?cons2 :min ?value) -> (?cons1 :min ?value)
]
```

Les deux règles ici, translator−consistency et translator−consistency−inverse, correspondent à une fonction de transformation et à son inverse. La première règle Jena stipule que la valeur maximale de la cohérence de l'entrée est appliquée à la sortie comme valeur maximale, alors que la fonction inverse applique la valeur minimale de la sortie comme valeur minimale possible pour l'entrée.

La règle de qualité Q2 est traduite en règle Jena comme ceci :

```
[translator-language-correctness:
(:translator :hasInput ?i), (:translator :hasOutput ?o),
(?i :createdByRule '1-2'), (?o :createdByRule '1-3'),
(?i :correct ?correct), (?o :consistency ?cons),
(?correct :hasValue ?correctValue),
equal(?correctValue, 'False')  -> (?cons :max '1')
]
```

Nous identifions le patron XPath ayant créé une donnée à l'aide d'un champ *createdByRule*. Ce champ comprend deux valeurs, la première permet d'identifier quelle

règle de provenance est liée à cette information (dans le cas de règles de provenance multiples pour un service), et la seconde permet d'identifier le patron XPath dans la règle lorsque plusieurs sont présents dans une règle de provenance.

Exemple 37

Considérons un service quelconque possédant deux règles de provenance :

$(E1)//A, //B \rightarrow //C$
$(E2)//D \rightarrow //E$

Ici, les nœuds retournés par le patron XPath $//A$ *sont identifiés à l'aide d'une valeur createdByRule égale à* $1-1$ *: première expression et premier patron de l'entrée. De la même manière, le patron* $//B$ *donne* $1-2$, $//C : 1-3$, $//D : 2-1$ *et* $//E : 2-2$.

Exemple 38

Avant inférence		
1-1 Consistency	1-2 Correct	1-3 Consistency
~~très haute~~	faux	très haute
~~haute~~		haute
moyenne		moyenne
basse		basse
très basse	faux	~~très basse~~

Q1,Q2 →

Après inférence		
1-1 Consistency	1-2 Correct	1-3 Consistency
~~très haute~~	vrai	**~~très haute~~**
~~haute~~		**~~haute~~**
moyenne		moyenne
basse		basse
~~très basse~~	faux	~~très basse~~

FIGURE 5.13 – Exemple application de qualité

Considérons par exemple les tableaux de la figure 5.13. Les valeurs de qualité sont un intervalle de cinq valeurs entre très basse et très haute. Le tableau de gauche contient les valeurs renseignées par un utilisateur, ou inférées par d'autres règles, et le tableau de droite les valeurs inférées par les règles (Q1) et (Q2).

Dans le tableau de gauche, les valeurs de cohérence de l'entrée $1-1$ *supérieures ou égales à haute sont désactivées (valeur maximale mise à moyenne), ainsi que la valeur de cohérence très basse de la sortie* $1-3$ *(valeur minimale mise à basse). L'application de la première règle de* $Q1$ *infère la désactivation des valeurs haute et très haute pour la cohérence de la sortie* $1-3$ *(la qualité de sortie doit être inférieure ou égale à celle de l'entrée). De la même manière, l'application de la seconde règle de* $Q1$ *infère la désactivation de la valeur très basse pour la cohérence de l'entrée* $1-3$. *Les valeurs concernant les données* $1-2$, *et la règle* $Q2$, *n'ont pas d'impact ici car aucune des deux valeurs (*$vrai$ *ou* $faux$*) n'a été modifiée par l'utilisateur.*

L'ajout de contraintes de qualité se fait par l'ajout de triplets dans la base de connaissance RDF. Lorsqu'un nouveau triplet "ex:A ex:predicate ex:C" est déduit à l'aide d'une règle, il est ajouté sans modification des autres triplets présents dans la base de connaissances. Aucune donnée n'est supprimée, facilitant la mise en place du système et évitant toute perte d'information en cas d'erreur, avec pour seule conséquence une augmentation de l'espace de stockage utilisé. Ceci permet également, dans les bases de graphes nommés, de garder une trace des modifications de qualité, et de créer un graphe de provenance des modifications de qualité.

Exemple 39

sujet	**prédicat**	**objet**
<execution>	:hasWorkflow	'workflowId'^^xsd :anyURI ;
<execution>	rdf :type	:traducteur
<execution>	:timestamp	'1234'^^xsd :integer ;
<execution>	:hasInput	<texte>
<texte>	:createdByRule	'1-1'
<execution>	:hasInput	<langage>
<langage>	:createdByRule	'1-2'
<execution>	:hasOutput	<traduction>
<traduction>	:createdByRule	'1-3'
<texte>	:consistency	<texte-cons>
<texte-cons>	:min	'0'^^xsd :integer ;
<texte-cons>	:max	'2'^^xsd :integer ;
<langage>	:correct	<correct>
<correct>	:hasValue	'False'^^xsd :boolean ;
<traduction>	:consistency	<traduction-cons>
<traduction-cons>	:min	'1'^^xsd :integer ;
<traduction-cons>	:max	'4'^^xsd :integer ;

FIGURE 5.14 – Exemple RDF

La figure 5.14 montre l'équivalent du tableau avant inférence dans le format RDF. Les valeurs discrètes y sont transformées en entier avec les valeurs : {très basse :0}, {basse :1}, {moyenne :2}, {haute :3}, {très haute :4}. Une valeur supplémentaire apparait dans ce tableau : la valeur de l'exactitude pour la langue, spécifiée à vrai. L'application de nouvelles contraintes de qualité se fait par l'ajout de triplets. Ainsi, l'application de la règle Jena translator-consistency ajoute la nouvelle contrainte :

<traduction-cons> :max '2'^^xsd :integer ;

L'application de la fonction inverse ajoute également un nouveau triplet :

<texte-cons> :min '1'^^xsd :integer ;

Les anciennes valeurs ne sont pas supprimées, il suffit lors de l'affichage de ne prendre que la valeur maximale des données min et la valeur minimale des données max.

La règle Jena translator − language − correctness, présente dans l'exemple 36 et correspondant à Q2, ajoute un nouveau triplet qui limite davantage la qualité de la donnée de sortie :

<traduction-cons> :max '1'^^xsd :integer ;

L'ensemble des informations de qualité étant contenu dans un triplestore et identifiable à l'aide d'un identifiant d'exécution de workflow, il est possible d'extraire un graphe de qualité à l'aide d'une requête SPARQL :

```
SELECT * WHERE {
?service :hasOutput ?output .
?service :hasInput ?input .
?service :hasWorkflow 'workflowID' .
?input :consistency ?consInput .
?consInput :min ?consIMin .
?consInput :max ?consIMax .
?output :consistency ?consOutput .
?consOutput :min ?consOMin .
?consOutput :max ?consOMax .
}
```

Cette requête retourne les entrées et sorties des services contenus dans une exécution de workflow $workflowID$, ainsi que l'intervalle de leur valeur de cohérence.

5.3 Architecture globale

Nous avons implémenté nos algorithmes et solutions comme une extension de la plateforme WebLab. Nous utilisons Apache Jena pour stocker et évaluer les règles d'inférence et Apache Tomcat pour la gestion des services web. Notre prototype est réalisé en Java, et intègre notre compilateur présenté dans la sous-section 5.1.4. L'architecture globale est visible sur la figure 5.15.

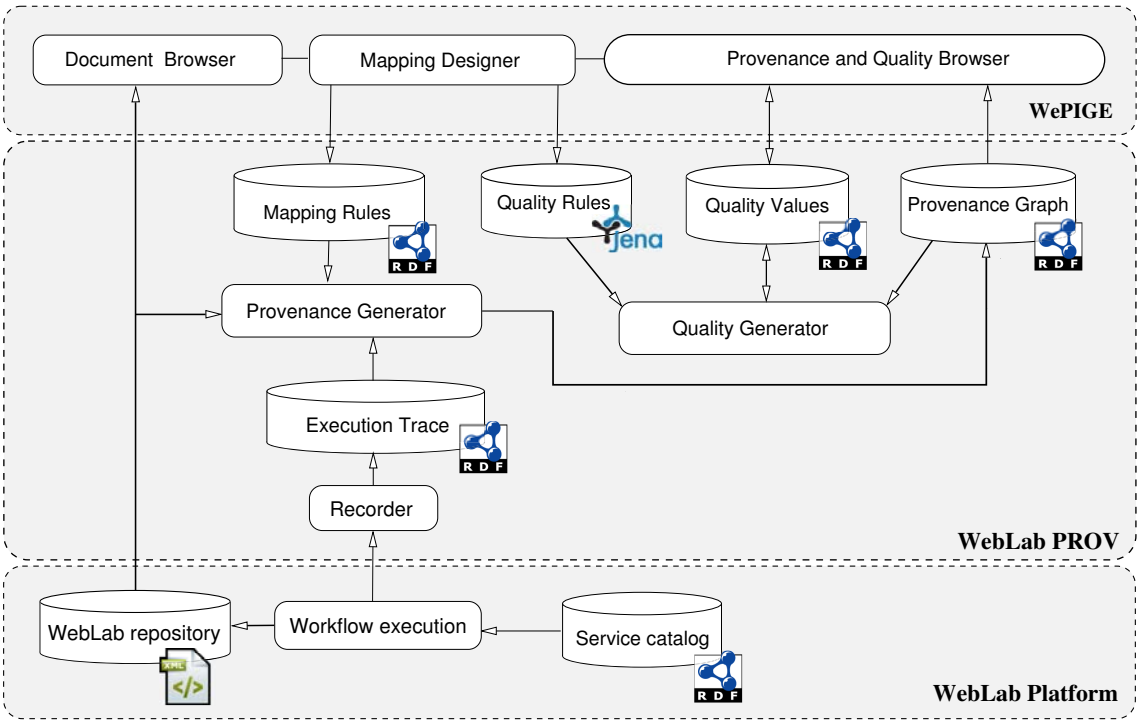


FIGURE 5.15 – Architecture globale

Nous pouvons voir sur l'architecture trois couches : la couche *plateforme WebLab*, la couche *WebLab Prov* et la couche *WePIGE*. La première contient les composants WebLab avec les données d'accès aux services dans le catalogue de services et le répertoire WebLab contenant les documents XML finaux. L'orchestrateur de services modifié fait appel à notre composant d'enregistrement (*Recorder*).

La couche *WebLab PROV* contient toutes les implémentations citées précédemment :

— l'enregistreur identifie les données importantes de l'exécution et les place dans le triplestore RDF.
— le générateur de provenance utilise les documents XML finaux, les règles de mapping, et les traces pour générer le graphe de provenance d'une exécution.
— le générateur de qualité utilise les règles de qualité, le graphe de provenance et les valeurs de qualité déjà connues pour inférer de nouvelles valeurs de qualité.

La dernière couche, *WePIGE* [18], correspond à l'interface graphique de notre application. Cette partie est composée de trois grands composants : l'explorateur de documents, le créateur de mapping, et l'explorateur de provenance et de qualité.

5.4 Interface utilisateur

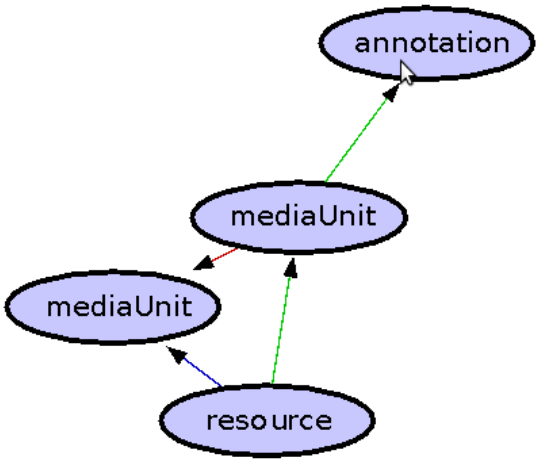


FIGURE 5.16 – Explorateur de provenance

La figure 5.16 montre le graphe de provenance lié à l'exécution d'un workflow. Différentes couleurs de flèche sont utilisées : une flèche rouge signifie un lien de provenance, une flèche bleue un lien de hiérarchie et une flèche verte les deux. Il est possible de filtrer les informations affichées à l'aide du panneau de gauche de l'application 5.17

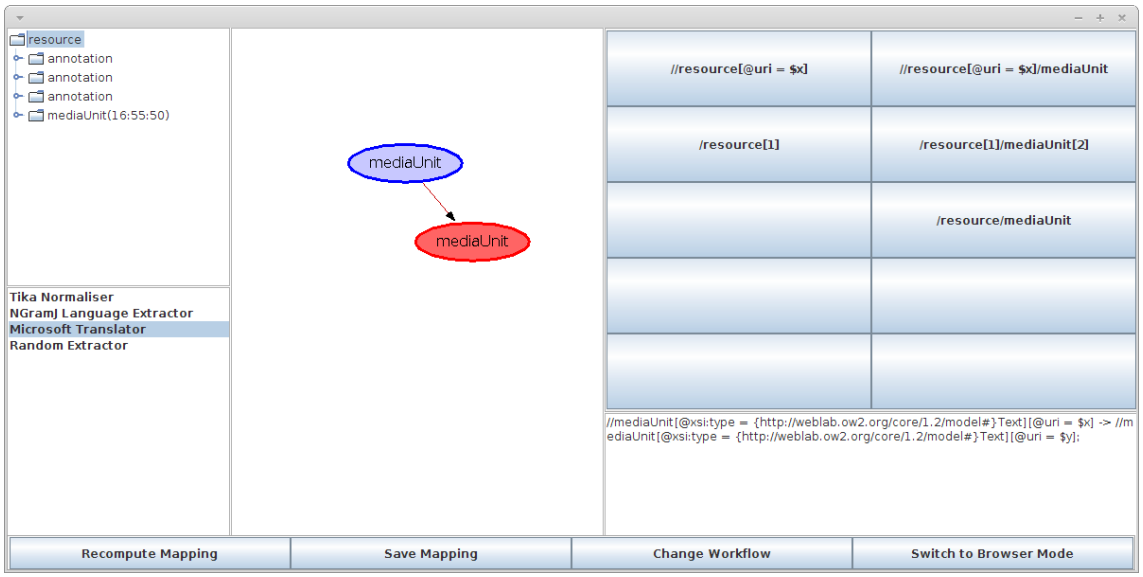


FIGURE 5.17 – Vue globale de l'application

L'explorateur de document, en haut à gauche de la figure 5.17, montre le document

XML sous la forme d'un arbre dont les nœuds peuvent afficher ou masquer leurs sous-nœuds pour une meilleure lisibilité et faciliter le parcours de l'arbre. En bas à gauche se trouvent les services exécutés au sein du workflow. La sélection d'un service permet de filtrer l'affichage dans le panneau de graphe de provenance. Ici, la sélection d'un traducteur, dont la règle de provenance est $//MediaUnit \rightarrow //MediaUnit$, permet de filtrer les deux nœuds présents sur le graphe. Sur le graphe, la ressource MediaUnit est entourée par une ligne bleue, tandis que la ressource sortante est cerclée de rouge. La partie droite de l'interface est le créateur de mapping.

Créateur de mapping

L'objectif principal de notre créateur de mapping [18] est d'inférer des règles de dépendance des données à partir d'un ensemble de nœuds sélectionnés. Les nœuds peuvent être sélectionnés dans l'explorateur de document pour les nœuds entrants, ou dans le graphe de provenance pour les deux types de nœuds (les nœuds passent en rouge lorsque sélectionnés). Deux ensembles sont considérés : l'ensemble I pour les nœuds d'entrée, et O pour les nœuds de sortie. Nous avons implémenté deux algorithmes.

Le premier algorithme est basé sur un algorithme d'apprentissage de requête [45]. Cet algorithme génère, pour les deux ensembles de nœuds sélectionnés I et O, les expressions communes XPath les plus précises $xp(I)$ and $xp(O)$ telles que $I \subseteq xp(I)(d)$ et $O \subseteq xp(O)(d)$. La mapping résultant est alors

$$xp(I) \rightarrow xp(O)$$

Le second algorithme est une extension du premier, mais ajoute des variables partagées entre les expressions XPath d'entrée et de sortie. L'algorithme calcule le plus proche ancêtre commun a de tous les nœuds présents dans les ensembles I et O. Si a est différent de la racine du document, alors il existe un préfixe commun $xp(a)$ de $xp(I)$ et $xp(O)$ tel que $xp(I) = xp(a)/xp_1$ et $xp(O) = xp(a)/xp_2$. Nous pouvons alors définir une nouvelle règle de dépendance des données

$$xp(a)[\$x = @uri]/xp_1 \rightarrow xp(a)[\$x = @uri]/xp_2$$

Tous les résultats sont alors montrés. L'utilisateur peut ensuite sélectionner deux patrons XPath, créant ainsi une règle affichée dans le champ texte en bas à gauche. Enfin, l'utilisateur peut y appliquer des modifications, puis l'exécuter pour afficher le résultat obtenu, ou la sauvegarder comme règle de provenance pour le service

éventuellement.

5.5 Conclusion

Nous avons montré dans ce chapitre une implémentation de notre modèle de provenance et de qualité dans le cadre de la plateforme WebLab. Nous avons utilisé les technologies liées à la plateforme WebLab telles que RDF, le langage d'inférence Apache Jena, XML, le langage de requête XQuery et Java, ainsi que des services WebLab existants. Cette implémentation montre la faisabilité de notre modèle dans le cadre de la plateforme WebLab.

6 Conclusion

6.1 Résumé

La gestion de la qualité des données dans les workflows pose de nombreux défis. Dans cette thèse, nous avons étudié le problème de l'annotation semi-automatique de données XML avec des valeurs de qualité.

Nous avons proposé une nouvelle approche fondée sur l'utilisation combinée de règles de provenance et de règles de qualité qui sont appliquées aux données et métadonnées produites par un workflow. L'avantage principal de cette approche est qu'elle est indépendante du modèle d'exécution du workflow et de la définition interne des services.

Cette approche a été conçue pour la plateforme WebLab, mais est assez générique pour être adaptée avec toute plateforme exécutant des workflows et dont les données sont enregistrées au format XML.

Nous avons également implémenté notre modèle au sein d'un prototype permettant aux utilisateurs de générer et de visionner la provenance et la qualité des données.

6.1.1 Modèle de Provenance WebLab

Notre première contribution est un modèle d'annotation de règles de provenance associées aux services WebLab permettant de générer les liens de dépendances entre les nœuds XML des documents WebLab. Ces règles sont décrites sous un format inspiré de XPath, et sont assez génériques pour être intégrées dans d'autres systèmes de workflows fondés sur l'échange de documents XML. Ce modèle ne demande aucune modification des services web exécutés, et est très peu invasif concernant l'orchestrateur de service.

6.1.2 Modèle de Qualité WebLab

Notre seconde contribution est un modèle de règles de qualité associées aux règles de provenance créées précédemment. Ces règles viennent en complément des outils d'analyse de la qualité existants (comme un analyseur orthographique par exemple). Ces règles relient une ou plusieurs entrées avec une ou plusieurs sorties, à l'aide d'opérateurs de comparaison, permettant d'effectuer des liens tels que : la *cohérence* du texte en entrée du service est toujours supérieure à la *cohérence* du texte en sortie. Elles permettent ainsi d'estimer la qualité d'une donnée, mais également de détecter des services qui ne fonctionnent pas comme prévu.

6.1.3 Réalisation

Enfin, nous avons implanté ces deux modèles dans une démarche d'intégration à la plateforme WebLab. Nous avons modifié l'orchestrateur de services (Java) afin de récolter des métadonnées d'exécution, et créé une grammaire (JavaCC) permettant de transformer nos règles en un langage intelligible par la machine (XQuery). Les données de provenance sont enregistrées dans un triplestore, au sein duquel nos règles de qualité, sous forme de règles d'inférence (Jena), permettent alors de créer les liens entre les dimensions de qualité. Nous avons également réalisé une IHM [1] facilitant à l'utilisateur la création de règles de provenance.

6.2 Perspectives

Nous avons présenté dans le chapitre 3 des fonctions de Skolem permettant de créer des règles d'agrégation, ainsi que des règles contextuelles permettant le fonctionnement du modèle dans le cadre de workflows non-linéaires (exécution de services en parallèle). Il serait intéressant d'étudier plus en profondeur les possibilités offertes par ces règles dans cadre de règles de provenance complexes.

Notre modèle actuel requiert un système fondé sur l'utilisation de documents XML pour stocker les informations. Il serait intéressant d'utiliser notre modèle dans d'autres systèmes applicatifs ou d'autres contextes. Nous pensons que le système est suffisamment générique pour être applicable dans des domaines très différents, tels que la sécurité, avec l'analyse des traces d'interactions utilisateurs.

Notre modèle actuel génère la provenance a posteriori des exécutions et propage automatiquement des métadonnées de qualité pour plusieurs exécutions du même workflow. Il devient alors possible d'analyser l'influence des composants utilisés sur la qualité des données et ainsi d'identifier les composants *sensibles*, et de proposer aux utilisateurs de modifier les workflows en conséquence.

Une dernière amélioration intéressante serait l'application de nos règles de provenance et de qualité au cours de l'exécution d'un workflow. Cela permettrait de générer les informations de provenance et de qualité de tous les services exécutés précédemment, fournissant de nouvelles métadonnées permettant de choisir dynamiquement le service suivant en fonction de la présence d'une donnée, ou de la qualité de celle-ci.

1. Interface Homme-Machine

Bibliographie

[1] Web Accessibility Initiative (WAI). http ://www.w3.org/WAI/.

[2] Parag Agrawal, Omar Benjelloun, Anish Das Sarma, Chris Hayworth, Shubha Nabar, Tomoe Sugihara, and Jennifer Widom. Trio : A system for data, uncertainty, and lineage. In VLDB, pages 1151–1154, 2006.

[3] Pinar Alper, Khalid Belhajjame, Carole A. Goble, and Pinar Karagoz. Enhancing and abstracting scientific workflow provenance for data publishing. In Proceedings of the Joint EDBT/ICDT 2013 Workshops, EDBT '13, pages 313–318, New York, NY, USA, 2013. ACM.

[4] Ilkay Altintas, Chad Berkley, Efrat Jaeger, Matthew Jones, Bertram Ludascher, and Steve Mock. Kepler : An extensible system for design and execution of scientific workflows. In SSDBM, pages 423–424, 2004.

[5] Yael Amsterdamer, Susan B. Davidson, Daniel Deutch, Tova Milo, Julia Stoyanovich, and Val Tannen. Putting lipstick on pig : enabling database-style workflow provenance. Proc. VLDB Endow., 5(4) :346–357, December 2011.

[6] Manish Kumar Anand, Shawn Bowers, and Bertram Ludäscher. Techniques for efficiently querying scientific workflow provenance graphs. In Proceedings of the 13th International Conference on Extending Database Technology, EDBT '10, pages 287–298, New York, NY, USA, 2010. ACM.

[7] Manish Kumar Anand, Shawn Bowers, Timothy McPhillips, and Bertram Ludäscher. Efficient provenance storage over nested data collections. In Proceedings of the 12th International Conference on Extending Database Technology : Advances in Database Technology, EDBT '09, pages 958–969, New York, NY, USA, 2009. ACM.

[8] Irit Askira Gelman and Anthony L. Barletta. A "Quick and Dirty" website data quality indicator. In WICOW'08 : Proceeding of the 2nd ACM workshop on Information Credibility On the Web, pages 43–46. ACM, October 2008.

[9] C. Batini and M. Scannapieca. Data quality : concepts, methodologies and techniques. Data-centric systems and applications. Springer, 2006.

[10] Khalid Belhajjame, Paolo Missier, and Carole A. Goble. Detecting duplicate records in scientific workflow results. In Proceedings of the 4th International Conference on Provenance and Annotation of Data and Processes, IPAW'12, pages 126–138, Berlin, Heidelberg, 2012. Springer-Verlag.

[11] Omar Benjelloun, Anish Das Sarma, Alon Halevy, and Jennifer Widom. Uldbs : Databases with uncertainty and lineage. In VLDB, 2006.

[12] Scott Boag, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie, and Jérôme Siméon. XQuery 1.0 : An XML query language. W3C Recommendation, January 2007.

[13] Peter Buneman, Adriane Chapman, and James Cheney. Provenance management in curated databases. In Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, SIGMOD '06, pages 539–550, New York, NY, USA, 2006. ACM.

[14] Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan. Why and where : A characterization of data provenance. In ICDT, pages 316–330, 2001.

[15] Peter Buneman and Wang-Chiew Tan. Provenance in databases. In Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data, SIGMOD '07, pages 1171–1173, New York, NY, USA, 2007. ACM.

[16] Mark H. Burstein, Jerry R. Hobbs, Ora Lassila, David Martin, Drew V. McDermott, Sheila A. McIlraith, Srini Narayanan, Massimo Paolucci, Terry R. Payne, and Katia P. Sycara. Daml-s : Web service description for the semantic web. In Proceedings of the First International Semantic Web Conference on The Semantic Web, ISWC '02, pages 348–363, London, UK, UK, 2002. Springer-Verlag.

[17] Steven P. Callahan, Juliana Freire, Emanuele Santos, Carlos E. Scheidegger, Cláudio T. Silva, and Huy T. Vo. Vistrails : visualization meets data management. In SIGMOD, pages 745–747, 2006.

[18] Clément Caron, Bernd Amann, Camélia Constantin, and Patrick Giroux. Wepige : The weblab provenance information generator and explorer. In EDBT, pages 664–667, 2014.

[19] Adriane Chapman, Barbara Blaustein, and Chris Elsaesser. Provenance-based belief. In Proceedings of the 2Nd Conference on Theory and Practice of Provenance, TAPP'10, pages 11–11, Berkeley, CA, USA, 2010. USENIX Association.

[20] James Cheney, Laura Chiticariu, and Wang-Chiew Tan. Provenance in databases : Why, how, and where. Found. Trends databases, 1(4) :379–474, April 2009.

[21] Reynold Cheng, Jinchuan Chen, and Xike Xie. Cleaning uncertain data with quality guarantees. VLDB Endow., 1(1) :722–735, 2008.

[22] Sarah Cohen-Boulakia, Olivier Biton, Shirley Cohen, and Susan Davidson. Addressing the provenance challenge using zoom. Concurrency and Computation : Practice and Experience, 20(5) :497–506, 2008.

[23] Y. Cui and J. Widom. Lineage tracing for general data warehouse transformations. The VLDB Journal, 12(1) :41–58, May 2003.

[24] Stéfan J. Darmoni, Vincent Leroux, Michel Daigne, Benoît Thirion, Pablo Santamaria, and Christophe Duvaux. Critères de qualité de l'information de santé sur l'Internet. Informatique et Santé, 10 :162–174, 1998.

[25] Susan B Davidson, Sarah Cohen Boulakia, Anat Eyal, Bertram Ludäscher, Timothy M McPhillips, Shawn Bowers, Manish Kumar Anand, and Juliana Freire. Provenance in scientific workflow systems. IEEE Data Eng. Bull., 30(4) :44–50, 2007.

[26] Susan B. Davidson and Juliana Freire. Provenance and scientific workflows : Challenges and opportunities. In SIGMOD, pages 1345–1350, 2008.

[27] Susan B. Davidson and Juliana Freire. Provenance and scientific workflows : challenges and opportunities. In Proceedings of the 2008 ACM SIGMOD international conference on Management of data, SIGMOD '08, pages 1345–1350, New York, NY, USA, 2008. ACM.

[28] Jérémie Doucy, Habib Abdulrab, Patrick Giroux, and Jean-Philippe Kotowicz. A new approach to populate a semantic service registry. In Proceedings of the 2010 international conference on Web information systems engineering, WISS'10, pages 112–125, Berlin, Heidelberg, 2011. Springer-Verlag.

[29] Gösta Grahne. Dependency satisfaction in databases with incomplete information. In Proceedings of the 10th International Conference on Very Large Data Bases, VLDB '84, pages 37–45, San Francisco, CA, USA, 1984. Morgan Kaufmann Publishers Inc.

[30] Todd J. Green, Grigoris Karvounarakis, Zachary G. Ives, and Val Tannen. Update exchange with mappings and provenance. In Proceedings of the 33rd International Conference on Very Large Data Bases, VLDB '07, pages 675–686. VLDB Endowment, 2007.

[31] Todd J. Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In Proceedings of the Twenty-sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS '07, pages 31–40, New York, NY, USA, 2007. ACM.

[32] Fabrice Guillet. Quality Measures in Data Mining. Springer-Verlag, Berlin, Heidelberg, 2009.

[33] Olaf Hartig and Jun Zhao. Using web data provenance for quality assessment. In In : Proc. of the Workshop on Semantic Web and Provenance Management at ISWC, 2009.

[34] RN Hook, M Romaniello, M Ullgrén, P Järveläinen, S Maisala, T Oittinen, V Savolainen, O Solin, J Tyynelä, M Peron, et al. Eso reflex : A graphical workflow engine for running recipes. In The 2007 ESO Instrument Calibration Workshop, pages 169–175. Springer, 2008.

[35] Duncan Hull, Katy Wolstencroft, Robert Stevens, Carole Goble, Mathew R. Pocock, Peter Li, and Tom Oinn. Taverna : a tool for building and running workflows of services. Nucleic acids research, 34(Web Server issue) :W729–W732, jul 2006.

[36] Tomasz Imieliński and Witold Lipski, Jr. Incomplete information in relational databases. J. ACM, 31(4) :761–791, 1984.

[37] VI Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. In Soviet Physics Doklady, volume 10, page 707, 1966.

[38] Yutu Liu, Anne H. Ngu, and Liang Z. Zeng. Qos computation and policing in dynamic web service selection. In Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters, WWW Alt. '04, pages 66–73, New York, NY, USA, 2004. ACM.

[39] Luc Moreau. The foundations for provenance on the web. Found. Trends Web Sci., 2(2–3) :99–241, February 2010.

[40] Luc Moreau, Juliana Freire, Joe Futrelle, RobertE. McGrath, Jim Myers, and Patrick Paulson. The open provenance model : An overview. In Juliana Freire, David Koop, and Luc Moreau, editors, Provenance and Annotation of Data and Processes, volume 5272 of Lecture Notes in Computer Science, pages 323–326. Springer Berlin Heidelberg, 2008.

[41] Kevin R Page, Benjamin Fields, Bart J Nagel, Gianni O'Neill, David C De Roure, and Tim Crawford. Semantics for music analysis through linked data : How country is my country ? In e-Science (e-Science), 2010 IEEE Sixth International Conference on, pages 41–48. IEEE, 2010.

[42] Barbara Pernici and Monica Scannapieco. Data quality in web information systems. In Proceedings of the 21st International Conference on Conceptual Modeling, ER '02, pages 397–413, London, UK, UK, 2002. Springer-Verlag.

[43] Leo L. Pipino, Yang W. Lee, and Richard Y. Wang. Data quality assessment. Commun. ACM, 45(4) :211–218, April 2002.

[44] Yogesh L. Simmhan, Beth Plale, and Dennis Gannon. A survey of data provenance in e-science. SIGMOD Rec., 34(3) :31–36, September 2005.

[45] Slawomir Staworko and Piotr Wieczorek. Learning Twig and Path Queries. In International Conference on Database Theory (ICDT), Berlin, Germany, March 2012.

[46] Robert D Stevens, Alan J Robinson, and Carole A Goble. mygrid : personalised bioinformatics on the information grid. Bioinformatics, 19(suppl 1) :i302–i304, 2003.

[47] Sreeni Viswanadha, Sriram Sankar, and al. Java compiler compiler (javacc) - the java parser generator. Sun 4.0 edition, January 2009.

[48] Richard Y. Wang and Diane M. Strong. Beyond accuracy : What data quality means to data consumers. J. Manage. Inf. Syst., 12(4) :5–33, March 1996.

[49] Zhibiao Wu and Martha Palmer. Verbs semantics and lexical selection. In Proceedings of the 32Nd Annual Meeting on Association for Computational Linguistics, ACL '94, pages 133–138, Stroudsburg, PA, USA, 1994. Association for Computational Linguistics.

[50] Tao Yu, Yue Zhang, and Kwei-Jay Lin. Efficient algorithms for web services selection with end-to-end qos constraints. ACM Trans. Web, 1, May 2007.

[51] Liangzhao Zeng, Boualem Benatallah, Anne H.H. Ngu, Marlon Dumas, Jayant Kalagnanam, and Henry Chang. Qos-aware middleware for web services composition. IEEE Trans. Softw. Eng., 30 :311–327, May 2004.

[52] Liangzhao Zeng, Hui Lei, and Henry Chang. Monitoring the qos for web services. In Bernd Krämer, Kwei-Jay Lin, and Priya Narasimhan, editors, Service-Oriented Computing – ICSOC 2007, volume 4749 of Lecture Notes

in Computer Science, pages 132–144. Springer Berlin / Heidelberg, 2007. 10.1007/978-3-540-74974-5_11.

[53] Huiyuan Zheng, Weiliang Zhao, Jian Yang, and Athman Bouguettaya. Qos analysis for web service composition. In Proceedings of the 2009 IEEE International Conference on Services Computing, SCC '09, pages 235–242, Washington, DC, USA, 2009. IEEE Computer Society.